Р. СКОТТ
М. СИНДАК

ПЛ·1 ДЛЯ ПРОГРАММИСТОВ

PL/1 FOR PROGRAMMERS

RAMON C. SCOTT,
NORMAN E. SONDAK

Worcester Polytechnic
Institute, Worcester,
Massachusetts

ADDISON—WESLEY PUBLISHING COMPANY
Reading, Massachusetts

Menlo Park, California — London — Don Mills, Ontario

СКОТТ Р.
СОНДАК Н.

ПЛ/1 ДЛЯ ПРОГРАММИСТОВ

Перевод с английского
Э. А. Трахтенгерца

МОСКВА «СТАТИСТИКА» 1977

Скотт Р. и Сондак Н.

С44 ПЛ/1 для программистов. Пер. с англ. Э. А. Трахтенгерца. М., «Статистика», 1977.

224 с. с ил.

ПЛ/1 — язык программирования, который можно применять для программирования широкого класса задач. В языке ПЛ/1 заложены возможности применении его для машин четвертого поколении. Данная книга представляет собой учебное пособие. Оно ориентировано на конкретную реализацию языка для системы IBM-360, транслятор которой может быть воспроизведен на советских машинах семейства ЕС.

Круг читателей — все, кто занимается математическим обеспечением ЭВМ, как практики, так и научные работники, а также учащиеся по данной специальности.

С $\frac{30502\text{-}053}{008(01)\text{-}77}$ 108-77 6Ф7.3

ПРЕДИСЛОВИЕ К РУССКОМУ ИЗДАНИЮ

Большой опыт эксплуатацнн электронных вычнслнтельных машнн показал, что для нх успешного применення необходнма снстема языков, обеспечнвающая запнсь алгорнтмов решення шнрокого класса задач.

В разработке такой снстемы языков наметилось трн тенденцнн. Стороннпкн первой счнтают, что основу снстемы должен составлять язык с тщательно отобранным и сравнительно небольшнм набором объектов н операцнй, допускающий возможности расшнрения и нмеющнй строго формалнзованные снntаксис н семантнку. В этом случае любая языковая снстема, орнентнрованная на решенне некоторого класса задач, должна представлять собой надстройку над базовым языком, образованную средствамн языка.

Стороннпкн второй тенденцнн счнтают, что должна существовать снстема языков специалнзнрованного н общего назначення. Пользователн прн напнсаннн сложных программ выбнрают те языки, на которых удобнее пнсать их отдельные части. В процессе трансляцнн программа, напнсанная на разлнчных языках, собнрается в еднную программу на внутреннем языке машнны.

Наконец, стороннпкн третьей тенденцнн пытаются создать едпный язык, который обладал бы большнм запасом конкретных нзобразнтельных средств, обеспечнвающнх опнсанне всех основных операцнй и объектов, употребляемых в задачах сегодняшнего дня.

Создателн языка ПЛ/1 — сотрудпнкн фнрмы IBM и Ассоцнацнн пользователей машннамн IBM — убежденные стороннпкн третьей тенденцнн. С момента своего возннкновення язык претерпел большие нзменення. Этн нзменения были вызваны как опытом применення языка н создання трансляторов, так н ожесточенной крнтикой его протнвнпков. К настоящему временн язык стабилнзнровался и прнобретает все бо́льшую популярность.

Язык ПЛ/1 появился после создания целой группы весьма совершенных языков и, конечио, оии оказали иа него большое влияние. Но иаряду с уже сложившимися идеями и понятиями ПЛ/1 вобрал в себя новые, такие, как привлечение широкого иабора типов данных и создание из них сложных структур, организация программной реакции на прерывания, возможиость организации параллельиых вычислений, преобразование программ в процессе траисляции и т. п. Эти особенности ПЛ/1 в сочетании с весьма совершенными траисляторами обеспечивают большие возможиости и удобства написания программ для решения широкого класса задач, включая программы, работающие в реальном масштабе времеии.

Книга представляет собой перевод руководства по программированию на ПЛ/1. В связи с тем что язык включен в математическое обеспечение ЕС ЭВМ и область его применения очеиь широка, книга Р. Скотта и Н. Соидака представляет интерес для круга специалистов, связаииых с примеиеиием вычислительной техиики.

Э. А. ТРАХТЕНГЕРЦ

ПРЕДИСЛОВИЕ

Задача книги — облегчить обучение программированию на языке ПЛ/1. Получивший широкое распространение язык программирования Фортран IV слишком ограничен и во многих случаях недостаточно удобен. Кроме того, при обучении технике программирования наряду со стандартными упражнениями на применение численных методов следует привлекать задачи на обработку текстов, списковых структур и операции над различными множествами символов. Для ясного понимания потенциальных возможностей вычислительных систем необходимо хорошо представлять себе методы машинной обработки символьной информации. Фортран не только затрудняет понимание этого аспекта отработки информации. Подчеркивая способность вычислительных машин быть только «гигантскими арифмометрами» и образуя вычислительный барьер на пути выражения рекурсивных зависимостей, он значительно ограничивает возможность применения ЭВМ.

Авторы настоящей книги давно поняли, что преподавание курса программирования в Вустерском Политехническом институте значительно улучшилось бы, если бы при этом можно было пользоваться языком, более развитым, чем Фортран. И когда фирма IBM начала вводить язык программирования ПЛ/1 для системы OS 360, этот язык привлек внимание авторов. Хотя мы были обескуражены трудностями, столь обычными на первоначальных стадиях работы с новым языком, он показался нам очень интересным с теоретической точки зрения.

Самым серьезным камнем преткновения в обучении новому языку было отсутствие учебных пособий.

Руководства составлялись и разрабатывались для других целей и не могли служить учебными пособиями. Одна из причин относительно медленного введения ПЛ/1 в программы высшей школы, несомненно, заключается в отсутствии подходящих учебников, снабженных хорошими примерами. Необходимость создания такого руководства и привела к написанию курса «ПЛ/1 для программистов».

Настоящая книга учитывает как требования учебных программ по современному программированию, так и нужды большого числа профессиональных программистов и создателей вычислительных систем.

В главе 1 излагается краткая история вычислительной техники и возникновения языка ПЛ/1.

В главе 2 дается краткое описание созданных до ПЛ/1 языков программирования. Это поможет читателю начать работать с новым языком. Глава построена так, чтобы после ее проработки студенты могли начать писать содержательные программы на ПЛ/1. Возможность написания программ на такой ранней стадии обучения — лучшее доказательство того, что языком ПЛ/1 довольно легко овладеть.

В главах 3—6 в значительной мере представлен синтаксис языка ПЛ/1. Эти главы снабжены необходимыми иллюстрациями и примерами с тем, чтобы читатель мог сразу же уловить многосторонние возможности языка.

При создании книги особенно большое внимание уделялось отбору примеров и иллюстраций, которые подчеркивают основные характеристики языка.

Авторы старались отобрать по возможности «долговечный» материал и избегать примеров, которые представляют интерес только для небольшой группы пользователей. В главе 7 даются наиболее сложные характеристики ПЛ/1.

Все программы, приведенные в книге, опробовались на системе IBM-360 (модель 40), а примеры отбирались в соответствии с практическими задачами по программированию.

Р. СКОТТ, Н. СОНДАК

Вустер, Массачусетс
Август 1969 г.

Глава 1

ПОЧЕМУ ПЛ/1?

1.1. ВВЕДЕНИЕ

Вычислительная работа, обеспечивающая деятельность человека, прошла эволюцию от счета при помощи пальцев руки до применения гигантских электронных систем по обработке данных. Большинство средств и усилий в области вычислительной техники сконцентрировано сейчас на подготовке программ для решения практических задач. Язык ПЛ/1 создан для того, чтобы сделать программирование более легким, более эффективным и более многосторонним. В этой главе будет кратко показано, как развитие вычислительной техники и математического обеспечения привело к созданию языка ПЛ/1 (Programming Language/One — язык программирования-один).

1.2. КРАТКАЯ ИСТОРИЯ СОЗДАНИЯ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Во все века люди стремились к созданию механического устройства, которое облегчало бы мыслительные процессы. Еще до создания системы чисел человек для облегчения счета прибегал к насечкам на костях, груде камней и собственным пальцам. Слово «digit»* — десятичная цифровая система — исторически уходит корнями в использование пальцев для счета. Такая примитивная «вычислительная машина» («digital computer») широко применима и в настоящее время. В зависимости от «решающих» устройств возникают и некоторые другие основы для систем счета. Например, некоторые примитивные племена, которые при счете пользуются суставами пальцев, фактически применяют троичную систему счисления (по числу суставов).

* В переводе означает «единица», «палец». — *Примеч. пер.*

Своеобразиые вычислнтельиые устройства были иайдены в руииах Древией Грецни, а само слово «calculus» (вычислять) относнтся к табуляциоиией системе, прииятой в Древием Риме. Каицелярские счеты, впервые появившиеся иа Востоке, представляют собой первое механическое вычислнтельиое устройство. До сих пор с нх помощью выполняются многне канцелярские операции.

К иачалу XVII в. было нзобретено иесколько устройств, помогающих счету. Одно из первых — палочки или «костн» Нэпера (1617 г.), которые позволялн проводнть умноженне. Примерно в то же время иескольким людьми одиовремеиио были создаиы различиые вариаиты устройства, известиого сейчас как логарифмическая лниейка.

Первой по-иастоящему счетной машнной можно назвать машииу Паскаля, нзобретеиную в 1642 г.,когда он был еще подростком. Это устройство с зубчатой передачей было построено нм для помощи отцу, правительствеииому ревизору фииаисов. В последующие годы Лейбиниц и другие усовершеиствовали устройство Паскаля н создали прообраз современных иастольиых вычислнтельных машин.

Наиболее смелый и блестящнй шаг на пути создаиня современных компьютеров был сделаи англичанином Чарльзом Бэббиджем в период между 1820 и 1856 г. Он поставнл перед собой цель сконструировать машину, которая могла бы полиостью решать сложиые математические задачн, производя иеобходимые арифметические действня, хранить полученные результаты, выполнять набор команд н даже печатать ответ. Ради этой иден Бэббидж отказался от должиости профессора на кафедре математики Кембриджского университета. Он вкладывал собственные средства в создание различиых автоматов. (Эти средства он, в конце концов, потерял.) Бэббндж действительио создал рабочую модель части одной из машин и получил субсидию от Британского правительства для конструирования «дифференцирующей машииы», которая оперировала бы 26-значнымн цифрами н печатала прираще́ния до шестого порядка. Потратив почти десять лет, Бэббидж не смог получить реальных результатов, и субсидия правительства была аииулироваиа.

Затем Бэббидж начал работать над «аналитнческой машиной», которая должиа была автоматически проводить серию арифметических действий в определениой псследовательиости. Но и ее он не смог довести до конца. Неудачи были вызваны не ошибками в коиструкцни, а несоответствием возможностей техники того времени замыслам Бэббиджа. Основные элементы, предложениые Бэббиджем, такие, как даиные и команды, вводимые в машину на перфокартах, условная передача управления, основанная на получеииых результатах, модификация команд самой машиной, были настолько хорошо разработаны, что когда была сконструирована первая электроииая вычислительная машииа, эти элементы были почти теми же, что и в «аналитической машине», скоиструированной Бэббиджем более ста лет тому назад.

В 1875 г. американец Стефен Болдуин получил патент иа создание устройства, с помощью которого можио было выполиять четыре арифметических действия, ие переиастраивая его. К началу второй мировой войиы Марчант, Барроуз н другие предложили ручиые н автоматиче-

скне настольные вычнслнтельные машины, которые по структуре напоминалн устройства, существующие в настоящее время. Умножение десятнзначных чнсел производилось за 6—10 с.

Примерно в то же время, что и Болдуин, Герман Холлернт в США изобрел машнну, работающую с примененнем перфокарт. Он сконструировал и получнл патент на запоминающее и анализирующее устройство для обработки демографнческих данных. Данные наносилнсь на перфокарты. Перепнсь 1890 г. обрабатывалась при помощи табуляторов Холлернта. Это дало выигрыш во временн на одну треть по сравнению с обработкой данных предыдущей переписи, и результаты обработкн былн более точными.

В 1911 г. была создана фнрма «Computing and Tabulating Recording Company», которая позже стала называться «International Business Machine Corporation» (IBM). Эта фирма взяла в свои руки контроль над основными патентами, полученными Холлеритом на перфорнрующие машины и другне устройства.

К 30-м годам стала очевидна связь между релейными схемами и булевой алгеброй (алгеброй логики). Из электромагнитных реле создавали логнческне схемы для машин, оперирующнх картами. Эти машины могли выполнять несколько довольно сложных арифметических действий.

Первым прямым наследннком «аналитической машины» Бэббиджа стал автоматическнй последовательный управляемый вычнслитель фнрмы IBM. Он был создан фирмой в 1944 г. совместно с Гарвардским университетом. Машнна была очень громоздкой, весила почти 5 тонн. В 1948 г. в Долгрене (штат Внргнння) была создана улучшенная модель этой машины.

Во время второй мировой войны теория электронных схем получнла дальнейшее развнтне. Необходнмость переработкн большого количества данных привела к созданию первой электронной счетно-решающей машнны Electronic Numerical Integrator and Calculator (ENIAC). Она состояла из трндцати блоков, занимающих свыше 15 000 квадратных футов, н весила более 30 тонн. В ее конструкции нспользовалось почти 19 000 вакуумных ламп. Она могла выполнять 5000 операцнй в секунду, а ее оператнвная память позволяла хранить 20 десятизначных цнфр.

Машнна ENIAC была создана под руководством доктора Джона В. Мочлн нз Пенснльванского университета и Дж. Преспера Эккерта, аспиранта этого уннверситета. Первой задачей, решенной на ENIAC, была задача по ядерной физике. Общее время ее решення составнло две недели, хотя самн вычнслення заняли всего два часа. Было подсчитано, что если бы при решенин прнменялнсь старые методы, то потребовалось бы не менее 100 человеко-лет.

Вслед за первой электронной вычнслнтельной машиной общего назначення появилось несколько других. Следующая машнна, которую сконструировалн Мочли и Эккерт (Binary Automatic Computer), была выполнена только в одном экземпляре. Несколько модификаций ее было создано другими различными фирмами.

Сейчас кажется странным, что в то время делались предсказания о необходнмостн для США иметь только девять машин подобного типа. Подсчнтано, что к семидесятым годам этого столетия в США будет работать не менее 80 000 вычнслительных машнн, а оперировать они будут порядками величин, гораздо большими, чем первая машина ENIAC.

Логическне схемы вычислительных машнн были разработаны в конце 40-х годов Джоном фон Нейманом, Германом Гольдстайном н А. В. Бёрксом. Матернал о них был опублнкован в серии статей, ныне широко нзвестных. Основная идея машины фон Неймана заключалась в том, что команды, так же как и данные, могут храниться в машине в цифровой форме и подвергаться однпаковым операциям модификации и обработки. Его машина была названа программно-управляемой счетно-решающей машиной. Для упрощения логическнх схем машин фон Нейман предлагал двончную снстему счисления.

В 1954 г. фирма IBM выпустила вычислительную машину модели 650. Это была первая программно-управляемая машина на вакуумных лампах и с запоминающим устройством на магнитном барабане. Вскоре были введены в эксплуатацию около 1000 машин этой модели. Библиотека программ к этой машине была самой дорогостоящей из всех, созданных ранее. Большая практнческая ценность этих электронпых вычислительных машин превратила их нз лабораторной диковины в рабочий ннструмент. Программы для модели 650 записывалнсь в кодах команд машины. Запись была очень трудоемким и утомительным делом. Часто допускались ошнбкн. Прнрода программ была ндеальна для того, чтобы их писали сами машнны. Высказывалась мысль о том, что если бы программы могла записывать машина, то эта тяжелая работа была бы значительно облегчена. Появилась ндея о созданин языков высокого уровня, на которых можно было бы писать программы для электронных вычислительных машин. Предполагалось, что машина должна была сама создавать программы, однако технические возможности ЭВМ еще не позволяли решнть эту задачу.

Примерно к тому времени, когда ЭВМ модели 650 достигла наибольшей популярности, относится создание трансляторов.

Конструкторы машин видели огромные возможности в повышении быстродействия и надежностн машин за счет замены вакуумных ламп полупроводниками. Прнмерно к тому же временн относнтся создание элементов быстродействующей ферритовой памяти. Эти элементы были более компактными и надежными и почти на три порядка более быстродействующими, чем запомннающие устройства на магнитных барабанах, прнменяемые в ЭВМ модели 650. Первой машнной, работающей на полупроводниках с ферритовой памятью, была модель RCA-501, поступившая на рынок в 1959 г. Тогда же было введено в эксплуатацию около 100 таких машин и некоторые из них работают до настоящего времени. Эта модель ЭВМ представляла «второе поколенне» вычнслительной техники.

Темп развития электронных вычислительных машин все возрастал. Новые поколення машин появлялись каждые пять лет (нормальным считается двадцатилетний цикл). В течение нескольких лет машнны первого поколения были почти полностью заменены машинами второго

поколения, обладающимн бóльшими возможностямн н мощностямн. В эксплуатацию вводнлись тысячн новых вычнслнтельных машнн.

Подчиняясь ускоряющемуся темпу развнтня вычнслнтельной техники, фирма IBM объявила в 1964 г. о запуске в серню машин IBM-360. Это третье поколенне машнн родилось прнмерно через пять лет после создання машнн на полупроводниках. Основной чертой вычнслительных машин третьего поколення, которые стали выпускать все ведущие фнрмы, была нх способность решать задачн, о возможности решения которых даже не подозревали какнх-нибудь десять лет назад. Успех был обусловлен заменой полупроводников нитегрнрованными схемами, созданнем сверхбыстродействующих запомнающнх устройств, снстемы модульных блоков и более мощных средств ввода-вывода.

На раннх ступенях развнтня электронной вычнслительной техникн машины были ориентированы на пакетную обработку данных, т. е. на последовательное решенне задач. По мере возрастання мощностей машин и требованнй к нним становнлось ясно, что для большинства пользователей такой метод обработкн данных не эффектнвен. Нанлучшим способом повышения эффективности работы вычнслнтельных центров оказалось совмещение во времени вычнсленнй с вводом и выводом данных. Обработка данных на удаленных вычислительных центрах стала называться обработкой данных на расстоянин, а эксплуатация вычнслнтельной машины многими пользователями, одновременно решающимн различные задачи, — снстемой с разделением времени. К середине 60-х годов технология соединения внешних устройств с машиной достигла той стадии, когда обработка данных на расстоянии большими вычнслнтельнымн снстемамн стала обычной. Ожидают, что к середние 70-х годов как снстемы с разделеннем временн, так и обработка данных на расстоянин будут стандартнымн методамн для проведения вычислительных работ. Обработка данных на расстоянии представляет собой ввод данных с отдельных внешних устройств через каналы связи (например, телефонные) и вывод данных непосредственно на отдаленные внешние устройства.

История последннх лет знает только одно столь же мощное достижение человеческого разума, которое некоторым образом способствовало созданию электронных вычнслнтельных машин — это атомная энергия. Оба эти изобретения преобразили наш мир.

1.3. АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНОЙ МАШИНЫ

Архнтектура вычнслительной машины — далеко не последнй вопрос в инженерном проекте. Он важен также для системного алгорнтмнста н программнста, поскольку во многнх случаях действнтельное решение вычнслнтельной задачи определяется архнтектурой машины н ресурсами, доступнымн программнсту. В вычнслнтельную машину данные вводятся в форме цнфр и символов. Эти данные обрабатываются н результаты выдаются либо на печатающее устройство в форме, удобной для прочтення человеком, лнбо на внешнее запомннающее устройство в форме, удобной для восприятня машиной. Эти основные операцни

и диктуют архитектуру машиины. Главиые компоиеиты простейшей вычислительиой машиины следующие:

1. Устройства ввода-вывода (УВВ).
2. Цеитральный процессор (ЦП).
3. Память.

С помощью устройств ввода-вывода осуществляется передача ииформации в цеитральный процессор и получеиие ииформации из иего.

Даиные для ввода в машиину готовятся иа устройствах подготовки даиных. С некоторых устройств (иапример, телетайпов) оии вводятся в машиину иепосредствеиио, с других (типа перфораторов) — с помощью перфокарт, перфолеит или в некоторых случаях с помощью специальных преобразователей, преобразующих аиалоговые сигиалы в цифровой код. Ввод может также производиться с других вычислительиых машии с помощью магнитиых лент или магиитиых дисков.

Подготовка ииформации с помощью устройств типа телетайпа или перфоратора — чрезвычайно трудоемкое и очень дорогостоящее дело. Даже опытиый перфораторщик может обработать 100—150 восьмидесятиколоиных перфокарт в час. Выразим скорость подготовки ииформации и ввода ее в символах в секуиду. Перфораторщик иаиосит ииформацию иа перфокарты со скоростью 3 символа в секунду. Стаидартное устройство чтеиия с перфокарт может обрабатывать около 1000 карт в минуту, или приблизительно 1300 символов в секуиду. Даиные с магиитиых леит могут считываться или записываться иа магнитиые ленты со скоростью от 60 000 до 120 000 символов в секуиду. С магнитиых дисков и магиитиых барабанов ииформация считывается со скоростью от 150 000 до 300 000 символов в секунду. Цифры, которые здесь приводятся, весьма приблизительны, так как сейчас существует миого устройств в каждом из перечисленных типов, которые работают с гораздо большей скоростью чем, указаио в этом параграфе.

Устройства вывода могут представлять информацию либо в печатной форме, либо в форме, позволяющей вводить ииформацию в другую вычислительиую машину. Коммерческие печатающие устройства в вычислительных системах средией мощиости могут печатать свыше 1000 строк в миуту, или около 2000 символов в секунду. Скорость вывода ииформации во виутреинем машиинном коде производителя, как правило, та же, что и скорость ввода. Интересиым устройством вывода является дисплей, который иногда иазывают катодио-лучевой трубкой. В иекоторые типы дисплеев ииформация может быть введеиа с помощью светового пера и в виде, аиалогичиом тому, который используют в графикопостроителях. На таких дисплеях могут воспроизводиться графики и рисуики, созданиые с помощью электроиной вычислительной машиины, в некоторых проектах — словесный вывод и слуховой ввод ииформации. Но в иастоящее время такие устройства единичны и очень дороги.

Важио зиать, какой объем информации может храниться на магнитиых леитах, дисках и барабанах. Плотиость записи иа стандартной магиитиой леите длиной в 2400 футов составляет 800 бит на дюйм. Даиные обычно записываются иа 2200 футах бобины, теоретически иа одиу бобину может быть записаио примерно 21 миллиои символов.

Практически же на бобину записывают только около 9 миллионов символов, так как на ленте между записями должны быть оставлены пустые промежутки, на которые информация заноситься не может. Обычный объем памяти магнитных дисков колеблется от 7 до 14 миллионов символов; объем памяти магнитных барабанов — примерно от 2 до 200 миллионов символов.

Центральный процессор состоит из двух функциональных частей. Одна из них выполняет арифметические и логические операции, другая осуществляет управление и определяет последовательность выполнения команд.

Поскольку информация передается в центральный процессор в форме последовательностей дискретных импульсов, процессор обладает сверхвысоким быстродействием.

Центральные процессоры в более ранних устройствах выполняли команды и арифметические действия за одну миллисекунду (одну тысячную долю секунды). Современные процессоры выполняют действия за микросекунду (одну миллионную долю секунды) и наносекунду (одну миллиардную долю секунды). Существуют экспериментальные процессоры, которые выполняют операции за пикосекунду (одну триллионную долю секунды). Пределом скорости работы электронных схем сейчас становится скорость света (примерно девять дюймов в наносекунду). Это означает, что элементы электронных схем должны располагаться очень близко друг к другу, иначе будет теряться быстродействие.

Разработка электронных вычислительных устройств становится сложной проблемой. Скорость передачи информации в ЦП может быть в тысячу раз большей, чем скорость передачи информации с устройств ввода-вывода. Для того чтобы быстродействующий центральный процессор мог работать с полной нагрузкой, необходимо осуществлять операции ввода-вывода. Одновременное выполнение нескольких работ называется мультипроцессорной работой*. Одновременное выполнение нескольких программ называется мультипрограммированием. Одной из особенностей ЦП третьего поколения является возможность мультипрограммирования.

Магнитная память представляет собой хранилище команд данных, доступных быстродействующему центральному процессору. Информация может быть введена в память и считана из нее за микро- или наносекунды. Обычно, хотя и не всегда, в любую ячейку памяти можно обратиться за одно и то же время. Быстродействующая память современной вычислительной машины может хранить примерно от нескольких тысяч до миллиона символов. Из-за технологических сложностей производства быстродействующая память, как правило, — самая дорогостоящая часть вычислительной системы. Написание программы, кото-

---

* Мультипроцессорной вычислительной системой называется система, состоящая из нескольких вычислительных процессоров, имеющих возможность работы с общей частью памяти. Работа этих процессоров по одновременному выполнению нескольких программ планируется специальным устройством или программой. См., например, Мультипроцессорные вычислительные системы. Под ред. А. Я. Хетагурова. М., «Энергия», 1971. — *Примеч. пер.*

рая могла бы поместиться в доступной памяти, часто оказывается проблемой, с которой постоянно сталкиваются ученые, работающие в области вычислительной техники. Большинство усилий, направленных на экономию памяти, приводит к увеличению времени ее работы.

1.4. ИСТОРИЯ ПРОГРАММИРОВАНИЯ

Алгоритмы. Электронная вычислительная машина предназначена для решения задач. Она может выполнить это только в том случае, когда в нее вводится совокупность точно сформулированных правил, которые позволяют решать данную задачу. В самом общем смысле алгоритмом можно назвать любое описание процесса решения задачи. Для специалиста по вычислительной технике алгоритм — это точное описание вычислительного процесса или процесса обработки данных. При использовании соответствующих данных этот процесс приводит к определенному результату за конечное время*. (Набор команд вычислительной машины, описывающий процесс выполнения алгоритма, называется программой.) Алгоритмы, которые обычно описываются в терминах вычислительных и логических операций или передач управления, фактически образуют последовательность преобразований входных данных. Например, один из алгоритмов для решения задачи с помощью ЭВМ может состоять из пяти шагов:

1) формулировка задачи, которая должна быть решена;
2) формулирование логической или математической модели данной задачи;
3) выражение этой модели в терминах определенной программы;
4) выполнение программы на ЭВМ;
5) проверка полученных результатов для того, чтобы убедиться в правильности решения.

Некоторые из этих шагов могут показаться очень простыми, особенно первый. И действительно, когда дело касается естественных наук, формулировка задачи, как правило, не представляет трудности. Формулировка и описание задачи, касающейся других сфер человеческой деятельности, — часто довольно трудное дело. На ЭВМ нельзя решить задачу, если проведен поверхностный анализ и неряшливо определены требования. В отличие от человека вычислительная машина сама не может исправить ошибку, если только это не заложено в ее программе. Для успешного решения задачи машиной все шаги алгоритма должны быть очень тщательно разработаны.

Составление алгоритма представляет собой творческий процесс. Так же как и произведения искусства, алгоритмы могут быть плохими и хорошими. Умение составлять алгоритмы приходит с опытом. Алгоритмы, которые даны в настоящей книге в качестве примеров и упражнений, могут служить базой для их оценки. Просмотрите их. Обратите внимание на их массовость, результативность и детерминиро-

* Точное определение понятия алгоритма можно найти, например, в книгах: Айзерман М. А. [и др.]. Логика, автоматы, алгоритмы. М., Физматгиз, 1963; Кринецкий Н. А. [и др.]. Программирование. М., Физматгиз, 1963. — *Примеч. пер.*

ванность. Это основные черты алгоритма. Массовость говорит о разнообразии входных данных, которые могут быть обработаны. Результативность означает, что алгоритм дает результаты через некоторое конечное число шагов. Детерминированность показывает, что алгоритм точен и не допускает двусмысленности.

Составление алгоритмов — основа программирования. В настоящее время ведутся широкие исследования в области анализа основных структур эффективных алгоритмов. Исследования показывают, что фаза проверки — самая решающая и именно на эту фазу нужно обращать особое внимание. Это касается не только начинающих программистов, но и профессионалов. Всякий раз, когда это возможно, проверяйте ввод, анализ и результаты. Как можно меньше полагайтесь на случай.

Машинный язык. Для выполнения алгоритма на вычислительной машине требуется программа. ЭВМ воспринимает задачу только в виде особого набора команд. Этот набор команд называется машинным языком. Любой алгоритм, который обрабатывается на электронной вычислительной машине, должен быть выражен на машинном языке данной ЭВМ. Машинная команда для ЭВМ обычно состоит из кода операции, адресов данных, хранящихся в памяти машины, и кода длины команды (для машин с переменной длиной слова). Фактический формат команды некоторых ЭВМ может быть очень сложным. Обычно считают, что даже для опытного программиста нужно несколько месяцев, чтобы овладеть машинным языком. Машинный язык содержит много специфических элементов и любая программа нормального размера состоит из большого числа отдельных команд. Проверка программы в связи с этим очень трудна. При этом совершенно обязательно точное знание операций, выполняемых электронной вычислительной машиной. Вследствие всего этого создание программ, написанных на машинном языке, требует много времени программиста, а отладка программы — много машинного времени. Не имея опыта в написании программ на машинном языке, невозможно даже представить себе, насколько трудоемок такой способ коммуникации человека и машины. Для облегчения этой задачи программисты стремились улучшить методы работы. Первым таким улучшением было создание мнемокодов.

Мнемокоды. Мнемокод должен быть по структуре достаточно близок к основному набору команд машины. Коды операций записываются в виде мнемонических сокращений, а адреса данных и команд могут быть выражены в форме символов. Запоминать и применять мнемонические коды команд легче, чем собственно машинные коды. Уменьшается также количество описок при кодировании. В мнемокоде пользуются символьной адресацией в отличие от цифрового кодирования в машинных языках, поэтому мнемокод значительно облегчает работу программиста.

Мнемокодная программа наносится на перфокарты или перфоленты и вводится в машину. Там она обрабатывается с помощью ассемблера. Ассемблер — это программа, которая воспринимает входную информацию, записанную на мнемокоде, и переводит ее в программу, записанную на машинном языке.

Мнемокоды имеются почтн для всех вычислительных машин и до сих пор широко применяются. Они позволяют осуществлять тесную связь программиста с машиной и могут давать очень эффективные машнинные программы. Возможности ассемблеров были расширены введеннем макрокоманд (макросов).

Макросы — это особые блоки команд, которые могут быть вставлены в программу для выполнення определенных операций. Это особенно ценно при выполнении операций ввода-вывода, когда часто повторяются одни и те же наборы команд.

В качестве примера программирования на машинном языке и мнемокоде рассмотрим программу, написанную для вычислительных машин IBM-1620. Эта программа выполняет довольно простую задачу. Она будет считывать число N с перфокарты, затем считывать еще N чисел, складывать эти числа, умножит сумму на три и затем полученный результат отперфорирует на перфокарте в форме с плавающей точкой.

```
INPUT   DSS   80                          02402 00080
OUTPUT  DAC   50, ANSWER =+. E+
                                          02483 00100
        DAC   30,                         02583 00060
MINUS   DAC   1.—.                        02643 00002
        DC    8.30000000                  02651 00008
THREE   DC    2.01                        02653 00002
COUNT   DC    2.0                         02655 00002
        DC    8.0                         02663 00008
SUM     DC    2.0                         02665 00002
START   RNCD  INPUT                       02666 36 02402 00500
        SF    INPUT                       02678 32 02402 00000
        TF    COUNT. INPUT+1              02690 26 02655 02403
A       RNCD  INPUT                       02702 36 02402 00500
        FADD  SUM. INPUT+9                02714 01 02665 02411
        SM    COUNT. 1. 10                02726 12 02655 000—1
        CM    COUNT. 0.10                 02738 14 02655 000—0
        BNE   A                           02750 47 02702 01200
        FMUL  SUM. THREE                  02762 03 02665 02653
        BNF   B. SUM                      02774 44 02798 02665
        TF    OUTPUT+42. MINUS            02786 26 02525 02643
B       TNF   OUTPUT+46. SUM              02798 73 02529 02665
        BNF   C. SUM—2                    02810 44 02834 02663
        TF    OUTPUT+20. MINUS            02822 26 02503 02643
C       TNF   OUTPUT+38. SUM—2            02834 73 02521 02663
        WACD  OUTPUT                      02846 39 02483 00400
        CALL  EXIT                        02858 49 00769 00000
        DEND  START                       02666
```

Приведенная программа была отпечатана ассемблером во время трансляции программы в машинные команды. Слева дан образец программы на мнемокоде, а справа — та же программа на машнннном языке. Первая колонка в обенх частях обозначает ячейку памятн. В левой частн адрес этой ячейкн обозначен символическн, в правой указан фактический адрес, определенный ассемблером. Команда FADD, SUM, INPUT+9 породнла машнннную команду 01 02665 02411, которая храннтся в ячейке памяти с адресом 02714. Во время выполнения программы эта команда складывает два чнсла с плавающей запятой. Первое нз этих чнсел храннтся в символической ячейке SUM, а другое —

в INPUT+9 (фактические адреса ячеек 02665 и 02411). Результат хранится в ячейке SUM (02665).

Хотя мнемокод представляет собой шаг вперед по сравнению с машинным кодом, но он тоже страдает рядом недостатков. Самый серьезный недостаток заключается в том, что мнемокод, созданный для определенной машины, пригоден только при работе с ней. Идеальным было бы такое положение, когда пользователь мог бы выполнить написанную программу на любой наиболее доступной и эффективной ЭВМ. К несчастью, программы, написанные на мнемокоде, применимы только к той машине, для которой они были созданы. Шагом на пути создания программ, не зависящих от определенной машины, было создание автокодов.

Автокоды. Автокоды были задуманы как особый набор команд для выполнения специализированной задачи. Этот псевдонабор команд обычно использовался для решения научных и инженерных задач. Для научных вычислений типична задача: $C = A \cdot B$. В мнемокоде для какой-нибудь вычислительной машины эта операция потребовала бы двух или более команд; в автокоде она могла быть закодирована как УМН A, B, C. Эта команда означала бы, что требуется умножить A на B и результат хранить в C. Один из наиболее известных систем автокодов является автокод Бэлл, разработанный Bell Telephone laboratories в штате Нью-Джерси. Созданная там система может применяться на целом ряде вычислительных машин, и программы, написанные для одной машины, могут быть выполнены на другой. Автокоды все еще самое легкое средство понимания того, как функционирует вычислительная машина, хотя их популярность последнее время падает из-за потери эффективности при выполнении работы. Поэтому, в то время как ассемблеры и трансляторы совершенствовались, уже прилагались усилия для создания более совершенного способа коммуникации с машиной посредством языков высокого уровня.

1.5. АЛГОРИТМИЧЕСКИЕ ЯЗЫКИ

Применение мнемокода и автокода имеет ряд недостатков: для написания программы любой степени сложности требуется большое число команд; на кодирование затрачивается длительное время; много машинного времени расходуется при отладке; трудно восстановить логику программы, если требуется провести модификацию или расширение первоначальной работы; программа, написанная для определенной машины, непригодна для работы на машине другой конструкции. Стремление преодолеть эти недостатки обусловило развитие нового подхода к написанию программ и послужило стимулом к созданию языков высокого уровня.

К идеальному языку высокого уровня предъявляются следующие четыре основных требования. Он должен способствовать полному использованию мощности машины, быть близок к естественному языку, на котором описывается задача, предоставлять возможность экономной записи задачи, позволять работать на любой из существующих электронных вычислительных машин. Первым достаточно удовлетво-

рительным языком высокого уровня был Фортран (сокращение от formula translator)*. Этот язык был создан для решения научных и инженерных задач, которые могут быть выражены непосредственно рядом арифметических и логических формул. Фортран имеет сейчас несколько вариантов и представляет собой очень мощный и хорошо развитый язык.

Электронные вычислительные машины могут реализовывать программы только на машинном языке. Поэтому любой язык высокого уровня должен обязательно переводиться на язык, доступный машине. Эта работа выполняется программой, которая называется транслятором. Транслятор сканирует один или несколько раз программу, написанную на языке высокого уровня, составляя окончательно программу на машинном языке, которая и выполняется на ЭВМ. Первоначальная программа, написанная на языке высокого уровня, называется исходной программой; окончательная программа, составленная на машинном языке, называется выходной (рабочей) программой. По существу, транслятор переводит исходные программы в выходные. Трансляторы могут быть двух типов. Транслятор первого типа полностью перерабатывает всю рабочую программу для ее последующей реализации. Транслятор второго типа переводит каждый оператор программы, написанный на языке высокого уровня, на машинный язык для немедленной реализации. Первый метод обеспечивает эффективный способ составления выходной программы. Второй позволяет проверять каждый оператор во время написания программы. Такой транслятор называется интерпретатором**.

Фортран, так же как и все другие языки программирования, состоит из символов своего алфавита, которые предназначены для написания предложений, называемых операторами. Оператор на Фортране может:

1) определять арифметические действия, которые необходимо выполнить;

2) давать информацию для определения последовательности шагов, требующихся во время выполнения программы;

3) указывать, какие операции ввода-вывода необходимы для ввода данных и воспроизведения ответов;

4) выполнять административные функции, такие, как резервирование памяти и формирование форматов данных при вводе-выводе.

Эффективность любой системы определяется в процессе ее эксплуатации, и, несмотря на некоторые недостатки, Фортран выполняет свою работу в высшей степени удовлетворительно. Трансляторы с Фортрана существуют почти для всех вычислительных машин средней и большой мощности***. Эти трансляторы различны по возможностям и степени сложности, но с помощью даже самых простых трансляторов можно решать очень сложные научные и инженерные задачи. Специалисты

---

* В переводе с английского означает «переводчик формул». — *Примеч. пер.*

** В настоящее время интерпретаторы широко применяются для работы в диалоговом режиме. — *Примеч. пер.*

*** В последнее время подобные трансляторы созданы для большинства мини- и даже некоторых микро-машин. — *Примеч. пер.*

по вычислительной технике указывали на недостатки Фортрана еще на ранней стадии его применения. Ограничения Фортрана начинаются со строгих правил написания операторов и заканчиваются невозможностью описания рекурсивных процессов*. Для преодоления этих трудностей и сближения структуры языка и описываемых им алгоритмов был создан язык программирования Алгол. Алгол впервые появился в Европе, но вскоре был принят как международный язык программирования. Основополагающим документом для Алгола была публикация Association of Computing Machinery**, названная Алгол-60. Алгол позволяет сжато выражать различные задачи, но широкие возможности языка затрудняют его полную реализацию.

Алгол обычно применяется как язык публикации для описания иллюстративных программ в некоторых научных и технических журналах***. Жаль, что когда Алгол возник, он не получил широкого применения. Это произошло главным образом потому, что состояние вычислительной техники в то время не способствовало принятию такого языка.

Еще одним вариантом алгоритмического языка был MAD (Michigan Algorithmic Decoder), разработанный в Мичиганском университете. По структуре это язык близкий к Фортрану, но у него есть ряд преимуществ по сравнению с ранними диалектами этого языка. MAD не получил реального коммерческого признания, несмотря на присущие ему достоинства.

1.6. ЯЗЫКИ ДЛЯ РЕШЕНИЯ ЭКОНОМИЧЕСКИХ ЗАДАЧ

Широкое проникновение электронной вычислительной техники в экономику привело к тому, что в этой области стал необходим такой же анализ требований к программированию, какой раньше был сделан для решения научных задач. На начальных стадиях применения вычислительной техники считали, что требования к решению экономических задач в корне отличаются от требований к решению научных задач. Файлы с записями экономических задач содержат сотни тысяч отдельных данных. Их нужно было проанализировать, протабулировать, сум-

---

* В Фортране запрещено рекурсивное обращение к подпрограммам. Это запрещение делает несколько менее удобным, но отнюдь не невозможным описание рекурсивных процессов. — *Примеч. пер.*

** Есть русский перевод: Алгоритмический язык Алгол-60. Пересмотренное сообщение. Перевод с английского под ред. Ершова А. П., Лаврова С. С., Шура-Бура М. Р. М., «Мир», 1965. — *Примеч. пер.*

*** В США до последнего времени Алгол-60 широко применялся главным образом как язык публикаций. Журнал «Communication of the ACM» в каждом номере публикует алгоритмы на Алголе-60. Однако некоторые американские фирмы в последнее время изменили свое отношение к Алголу настолько, что начали закладывать в аппаратуру вычислительных машин средства, обеспечивающие эффективную трансляцию программ, написанных на Алголе. В Европе Алгол как язык программирования применяется довольно широко (см., например, Harder E. L. The Expanding World of Computers. Communication of the ACM, 1968, № 4). — *Примеч. пер.*

мировать и разбить на определенные календарные сроки. Хотя сложность арифметических вычислений при решении экономических задач сравнительно невелика (с точки зрения программирования), требовалось нечто более совершенное, чем мнемокод.

Первым шагом к созданию более совершенных методов программирования для пользователей электронных вычислительных машин было создание в коммерческих целях систем генераторов отчетов (Report Program Generator, RPG)*. Эти системы могли прочесть файл, переработать требуемую информацию и выдать на печать сообщение для анализа. При этом требовались детальная спецификация вводимого файла (включающая информацию о различных записях и элементах данных в файле), информация о данных вывода (включающая спецификацию типа вывода — печатные сообщения, итоговые карты и т. п.) и редактирование. Генераторы отчетов могли производить простейшие вычисления (например, суммирование).

Генераторы отчетов стали сразу популярным инструментом программирования экономических задач. Они быстро и эффективно производили однообразную работу и могли решать сравнительно простые задачи. Поскольку овладеть генераторами отсчетов было легко, даже неопытный программист мог пользоваться ЭВМ без длительной предварительной подготовки. Основной недостаток генераторов отчетов заключается в том, что они слишком ориентированы на какой-либо один тип машины и поэтому непригодны для других систем. Но, несмотря на этот недостаток, генераторы отчетов представляют собой очень эффективное средство для решения довольно широкого круга экономических задач.

Правительство США поняло, что для более эффективной обработки и решения экономических задач необходимо более независимое и общее средство программирования, и к 1960 г. был создан язык программирования экономических задач — Кобол (Common Business Oriented Language, COBOL)**. По идее Кобол должен был позволять писать программы на языке, близком к английскому, так, чтобы их можно было применять на машинах различных типов.

Необходимость обработки разнообразных форматов входных и выходных данных привела к тому, что транслятор Кобола усложнился по сравнению с транслятором Фортрана.

Программы на Коболе можно успешно переносить с одного типа вычислительных машин на другой, почти не внося никаких изменений.

Кобол состоит из четырех частей: раздела идентификаций, раздела оборудования, раздела данных и раздела процедур. Каждый из этих разделов выполняет специальную функцию. В разделе идентификации содержится описательная информация о транслируемой программе. Раздел оборудования сообщает транслятору характеристики и конфи-

---

* Описание этой системы можно найти в кн.: Фишер Ф. П. и Суиидл Д. Ф. Системы программирования. М., «Статистика», 1971. — *Примеч. пер.*

** Описание языка Кобол дано в кн.: Сэксон Дж. Кобол. М., «Статистика», 1970; Джермейн К. Программирование на IBM/360. М., «Мир», 1971. — *Примеч. пер.*

гурацию машины, на которой данная программа будет реализована. Это предоставляет возможность написания одной и той же программы на исходном языке для машин различных типов. Раздел данных дает характеристики каждого файла и предъявляет требования к памяти для конкретной программы. Раздел процедур описывает шаги, необходимые для реализации программы. Эти шаги записываются в виде операторов и могут быть представлены как предложения, параграфы и секции. Операторы делятся на повелительные, условные и директивы транслятору. Повелительные операторы предназначены для описания безусловных действий. Условные операторы позволяют изменять выполнение программы в зависимости от полученных результатов. Операторы директив транслятору, как показывает само название, управляют процессом трансляции.

Операторы Кобола состоят из набора символов, в них применяются знаки препинания и содержатся отдельные слова. Это делает их похожими на английские предложения. Слова в операторах представляют собой имена существительные и глаголы. Операторы могут также включать арифметические операции. Определенные слова имеют особое значение для транслятора, поэтому их применение ограничено. Обычно форма языка по структуре такова, что исходную программу сравнительно легко прочитать и определить, какие операции будут выполняться машиной.

Первый транслятор с Кобола подвергся критике, так как он не позволял составить достаточно эффективную рабочую программу. Более поздние варианты транслятора были значительно улучшены и расширены. Это сделало Кобол самым подходящим языком для решения коммерческих задач. Однако транслятор с Кобола не позволяет экономистам полностью использовать возможности вычислительных машин, а применение Фортрана не позволяет решать задачи с большим разнообразием структур файлов, что характерно для коммерческих задач. Потребовался новый подход к языку высокого уровня, так как коммерческие задачи часто приобретали черты научных задач, а научные задачи — черты экономических.

Изучение недостатков машинно-ориентированных языков выявило границы возможностей человека-оператора в управлении комплексом вычислительных машин. Машины становились все более производительными и нельзя было допускать перерывов в работе, пока оператор не выполнит какие-то отдельные операции. Для управления решениями отдельных задач и сокращения времени на отладку были созданы мониторы. Но, в конце концов, стало ясно, что независимо от степени квалификации оператор действовал слишком медленно и допускал ошибки во время непосредственной работы на вычислительной машине. Так созрела идея создания и применения операционных систем. Программа операционной системы частично хранится в оперативной памяти вычислительной машины с целью максимального использования всех ресурсов ЭВМ. Достаточно развитые операционные системы позволяют осуществлять непрерывное последовательное выполнение задач, обращаться к необходимым данным и подпрограммам, требующимся выполняемой программе. Если позволяют ресурсы, операционная систе-

ма выполняет одновременно две или более работы. Кроме того, операционная система учитывает машинное время. Это дает возможность предъявить пользователю точный счет за проделанную работу. Для управления работой вычислительной машины в операционные системы обычно включают как их составную часть трансляторы с различных языков высокого уровня.

В процессе применения операционных систем возник новый тип программистов, которых стали называть системными программистами. Системные программисты создают операционные системы, трансляторы и другие системы программ, позволяющие удобнее и эффективнее применять ЭВМ. Большинство системных программистов вынуждено было прибегать в своей работе к мнемокодам. Мнемокоды имели много недостатков, поэтому созрела необходимость создания более приемлемого языка программирования.

1.7. ВОЗНИКНОВЕНИЕ ЯЗЫКА ПРОГРАММИРОВАНИЯ ПЛ/1

В 1963 г. фирмой IBM был создан комитет представителей двух организаций: пользователей вычислительных машин для решения научных задач (SHARE) и пользователей машин для решения коммерческих задач (GUIDE). Этот комитет пришел к выводу, что для лучшего использования возрастающей мощности вычислительных систем необходимо разработать новый язык программирования высокого уровня. Этот новый язык, названный ПЛ/1, должен был вобрать в себя лучшие черты существующих языков и позволить полностью реализовывать потенциальные возможности современных вычислительных комплексов. Он должен был предоставить возможность решать научные задачи более разнообразные, чем позволял Фортран, и помочь программистам, решающим коммерческие задачи, в серьезных вычислениях, которые невозможны с помощью Кобола. Наконец, новый язык должен был дать системным программистам средство для решения задач в реальном масштабе времени, а также разработки эффективных операций ввода-вывода при соответствующих устройствах связи.

При написании программ на языке с такими широкими возможностями, как у ПЛ/1, отпадает необходимость создания программ на других языках. Применение только одного языка существенно уменьшает расходы на обучение программистов и упрощает проблемы стандартизации. Программисты, разрабатывающие программы для научных и коммерческих задач, и системные программисты могут легко понять друг друга. Они могут читать программы, написанные другими, обогащать друг друга своим опытом и знаниями и более эффективно решать стоящие перед ними задачи.

Создание языка ПЛ/1 преследовало также цель более эффективного использования возможностей современной вычислительной техники. Одна из наиболее важных черт вычислительных машин третьего поколения заключается в широком применении систем прерывания. Это устройство позволяет работать в режиме мультипрограммирования при переключении машины на решение другой задачи, когда по той или

ииой причиие необходимо прервать выполиеиие первой. Язык ПЛ/1, кроме того, легко связать с операциоииой системой современиых ЭВМ. Раиее создаииые языки ие давали такой возможиости пользоваться преимуществами вычислительиой техиики третьего поколеиия.

Одиим из иедостатков создаииых раиее языков высокого уровия были жесткие ограиичеиия на включеиие определеииых символов и примеиеиие правил иаписаиия программ. ПЛ/1 позволяет сиять эти ограиичеиия. Еще одио достоииство языка ПЛ/1 заключается в том, что пользователь может работать только с теми средствами языка, которые иужиы для решеиия его задачи. При этом ои ие обязаи зиать все средства языка. ПЛ/1 — достаточио простой язык, чтобы им мог овладеть начииающий программист, и достаточио мощиый, чтобы удовлетворять иужды квалифицироваииых пользователей.

Все сказаииое говорит о том, что новый язык высокого уровия ПЛ/1 мог бы стать едиистваииым языком программироваиия для реализации разиообразиых возможиостей любых вычислительиых систем. Ои мог бы позволить эффективио работать на ЭВМ различиой коифигурации и управлять устройствами ввода-вывода. К преимуществам языка ПЛ/1 можно отиести:

1) улучшеиие связи и ассимиляции разнообразиых программ для различиых прикладиых задач;

2) одииаковое использоваиие файлов в различиых прикладиых задачах;

3) миниимизацию иеобходимости применеиия машиииио-ориеитироваииых языков;

4) обеспечеиие лучшей подготовки программистов, поскольку требуется овладеть только одиим языком;

5) упрощеиие стаидартов разработки программ, их описаиия и отладки;

6) более полиое использоваиие ресурсов вычислительиой машиины за счет более эффективиого траислятора и программ, написаииых иа языке ПЛ/1.

Как всякий новый язык, ПЛ/1 ие избежал болезией роста. Так, до 1967 г. ие было версии траислятора, который можио было бы широко примеиять. В иачале 1968 г. создали улучшеииый вариаит трансляторa и тогда стали очевидными достоииства языка ПЛ/1. Одиой из самых строгих проверок языка является его способиость служить осиовой для создаиия другого языка. ПЛ/1 стал осиовой иового языка FORMAC.

В иастоящее время ПЛ/1 примеияется во миогих коммерческих фирмах для решеиия экоиомических задач и для обработки деловой информации. Человек, заиимающийся гумаиитариыми иауками и искусством, может воспользоваться языком ПЛ/1 как прекрасиым ииструмеитом для аиализа и изучеиия произведеиий литературы, музыки и изящиых искусств. Программист, заиимающийся решеиием иаучиых задач, иайдет в ПЛ/1 мощиое средство для решения аиалитических задач и задач по моделироваиию.

Глава 2

ПЛ/1 — БЫСТРОЕ НАЧАЛО

2.1. ВВЕДЕНИЕ

Один из лучших способов овладения новым языком — это применение его на практике. В данной главе будут рассмотрены только основные черты языка ПЛ/1, которые дадут пользователю возможность начать составлять простые программы. Для простоты и ясности введения в нем будут рассмотрены характеристики ПЛ/1, делающие его похожим на языки программирования типа Фортрана и Кобола для решения научных и экономических задач.

Систематическое изложение всех особенностей ПЛ/1 начнется в главе 3 и только тогда станут очевидными все возможности широкого применения этого языка. Примеры в книге иллюстрируют новые идеи, заложенные в ПЛ/1, и показывают, как использованы в нем старые. Непосредственное применение языка в реальных программах — самый убедительный способ перехода от теории к практике.

Авторы должны предупредить читателя, что в этой главе будут приведены только наиболее яркие примеры и показаны лишь некоторые возможности применения языка ПЛ/1. Некоторые конструкции языка, рассмотренные в настоящей главе, очень специфичны. Их мы включили с целью помочь обучающемуся сравнительно быстро овладеть навыком составления программ средней трудности.

Как указывалось в главе 1, программа, написанная на языке ПЛ/1, называется исходной программой. Для введения в машину она обычно перфорируется на карты. Для последующей обработки транслятор ПЛ/1 переводит ее на машинный язык. Программа, написанная на машинном языке, называется выходной или рабочей программой. В настоящее время существуют два варианта транслятора ПЛ/1. Транслятор, который позволяет реализовать все возможности языка, является частью операционной системы уровня F для машины IBM-360. Сущест-

вует также транслятор только для подмножества языка ПЛ/1. В настоящей кннге язык программнрования ПЛ/1 будет рассмотрен в полном объеме. Будут также рассмотрены огранпчення при использовании транслятора ленточной и дисковой операцнонных систем машины IBM-360. Таким образом, программист может сам решить, какой нз вариантов транслятора ПЛ/1 лучше применить.

За те годы, в течение которых Фортран и Кобол из довольно простых языков превратнлнсь в современные системы, все вносимые в них изменения заключалнсь во введении новых конструкций; никаких изменений ранее созданных конструкций не делалось. Одна из причнн такого положення заключалась в экономнческой стороне вопроса. Было жаль средств, вложенных в созданне программ и подпрограмм. Другой причиной была привычка; программнст, раз начав пользоваться новым вариантом языка, старался добавлять новые конструкцин только в случае необходимости.

В результате изложенного многие конструкцин, например в Фортране, выполняют двойные функции, и они могли бы использоваться гораздо полнее. Пожалуй, наиболее ясно эта избыточность проявляется в операторах REAL, COMPLEX, INTEGER, LOGICAL, DOUBLE PRECISION, DIMENSION, DATA, COMMON и EQUIVALENCE. Эти операторы предназначены для введення в транслятор информации о переменных, имеющнхся в программе, н выполняемые ими функцни перекрывают друг друга. Если взглянуть на проблему преобразования информации транслятором с чисто лингвистической точки зрения, то можно было бы избрать более простой метод. Создатели ПЛ/1 именно так подошлн к проблеме и поэтому смогли объединить функции, выполняемые этими операторами, в один оператор DECLARE (ОБЪЯВИТЬ). Оператор DECLARE, построенный на рациональной основе организации языка, может иметь гораздо более широкое применение, чем просто опнсанне данных, которое он выполнял в ранее созданных языках.

ПЛ/I — новый язык для программистов, но он базируется на уже накопленном опыте решення задач. Удивительно то, что трудности описания логики программ, свойственные ранее созданным языкам, исчезают, когда программа запнсывается на ПЛ/1. У большинства программистов есть своеобразный «спнсок пожеланий» — перечень того, что им хотелось бы выполнять непосредственно. Хотя ПЛ/1 еще не может удовлетворнть всех чаяний программистов, он снимает много вопросов из этого «списка пожеланий». Это делает ПЛ/1 одним нз наиболее важных, значительных и разносторонних языков, созданных в последнее десятилетне н отвечающих современному состоянию вычислительной техники.

Далее приводнтся законченная программа, записанная на языке ПЛ/1. Она нллюстрнрует некоторые особенности языка. Программа считывает с перфокарты данных целое число, отперфорированное на ней, определяя количество последующих карт данных. На каждой из последующих карт в колонках 1—40 перфорируется одно из имен в форме: SMITH, ROBERT J, а затем в колонках 45—80 семь однонлн двузначных чнсел, представляющих собой номера счетов этих

```
        /* CHAPTER # 2—EXAMPLE # 1 */
        /* SORTING NUMERICALLY AND ALPHABETICALLY */
 1 E201: PROCEDURE OPTIONS (MAIN);
        /* THE USE OF CONTROLLED DATA IN THE DECLARE STATEMENT (ABBREVIATED DCL)
        ALLOWS THE PROGRAMMER TO ALLOCATE STORAGE DURING EXECUTION

        OF THE PROGRAM. THE EXTERNAL ATTRIBUTE ELIMINATES THE NECESSITY OF PASSING DATA TO THE
        SUBROUTINE PROCEDURES VIA ARGUMENT LISTS. THIS IS SIMILAR TO THE COMMON
        STATEMENT IN FORTRAN AND IS DISCUSSED IN DETAIL IN CHAPTER SIX.  */
 2      DCL DATA___TYPE CHAR (7) INITIAL ('NUMERIC'), (NAMES (*) CHAR (40), SCORES (*)
          DEC FIXED (2)) CONTROLLED EXTERNAL, SCORE (7) DEC FIXED (2);
 3      GET LIST (NN);
        /* THE ALLOCATE STATEMENT DEFINES THE STORAGE REQJIREMENTS FOR CONTROLLED DATA
        DURING EXECUTION. */
 4      ALLOCATE NAMES (NN), SCORES (NN);
         /*  FIRST READ IN THE NAMES AND THE SCORES, THEN SCAN THE SCORES FOR THE HIGHEST
           AND LOWEST, AND THEN COMPUTE THE AVERAGE SCORE.     */
 5 DO I=I TO NN;
 6       GET EDIT (NAMES (I)) (COLUMN (1), A (40));
 7            DO J=1 TO 7;
 8                 GET LIST (SCORE (J));
 9            END;
10       MAXS, MINS=SCORE (1);
11       DO J=2 TO 7;
12            MAXS=MAX (SCORE (J), MAXS);
13            MINS=MIN (SCORE (J), MINS);
14       END;
15       SCORES (I)=(SUM (SCORE)—MINS—MAXS)/5;
16  END;
        /* NOTE THAT THE DO LOOP USED IN STATEMENT 17 IS DIFFERENT IN FORM THAN THAT IN STATEMENTS
           5,7 AND 11. PL/1 ALLOWS BOTH TYPES WITH SEVER EXTENSIONS, A VERY USEFUL ASSET TO THE
           PROGRAMMER    */
17  DO J=1,2;
        /* A CALL STATEMENT PRODUCES AN UNCONDITIONAL BRANCH TO ANOTHER PROCEDURE——HERE AN
           EXTERNAL ONE——THE VARIABLE DATA___TYPE, DECLARED TO BE A CHARACTER STRING OF LENGTH
           SIX, IS BEING PASSED TO THE PROCEDURE SORT.    */
18            CALL SORT (DATA___TYPE);
19            IF J=I THEN PUT EDIT ('TABLE OF SCORES FROM HIGHEST TO LOWEST')
                                  (SKIP (3), X (4), A);
```

```
21                ELSE PUT EDIT ('TABLE OF SCORES IN ALPHABETIC ORDER')
                                (SKIP (5), X (5), A);
22              PUT SKIP (2) EDIT ('NAME', 'SCORE') (X (18), A (22), A);
23              PUT SKIP EDIT( (NAMES (K), SCORES (K) DO K=1 TO NN)
                        (SKIP (1), A, 40), F (5) );
24              END;
25              PUT EDIT ('THE AVERAGE SCORE IS' , (SUM (SCORES)/NN)
                        (SKIP (5), X (5), A, F (3) );
26  END E201;
 1  SORT: PROCEDURE (DATA___TYPE) OPTIONS (MAIN);
    /*          THE USE OF THE '*' FOR A DIMENSION CAUSES THE VARIABLE ARRAY TO BE DIMENSIONED
                WHAT IT WAS IN THE LAST ALLOCATE STATEMENT.     */
 2        DCL DATA___TYPE CHAR(7), INDICATOR BIT (1), (NAMES (*) CHAR (40),
                        SCORES (*) DEC FIXED (2) ) CONTROLLED EXTERNAL, NAME CHAR (40);
 3        NN=DIM (NAMES, 1);
 4        IF DATA___TYPE='ALPHA' THEN CALL ALPHA;
 6                                ELSE CALL NUMERIC;
    /*          THE INTERNAL PROCEDURE NUMERIC SORTS THE SCORES IN DECENDING ORDER     */
 7                  NUMERIC: PROCEDURE;
 8                       GT=-1;
 9                  SORT: INDICATOR='0'B;    J=1; GT=GT+1;
12                       SCAN: J=J+1;
13                              IF SCORES (J) > SCORES (J-1) THEN CALL SWITCH;
15                              IF J < NN-GT THEN GO TO SCAN;
17                       IF INDICATOR='1'B THEN GO TO SORT;
19                  DATA___TYPE='ALPHA';
20                  END NUMERIC;
    /*          THE INTERNAL PROCEDURE ALPHA SORTS THE NAMES IN ALPHABETIC ORDER  */
21              ALPHA: PROCEDURE;
22                       GT=-1;
```

```
gg 23                SORT: INDICATOR='0'B;    J=1; CT=CT+1;
   26                     SCAN: J=J+1;        K=1;
   28                          TEST: IF SUBSTR (NAMES (J-1), K, 1)=='.' THEN DO;
   30                               IF SUBSTR (NAMES (J), K, 1)=SUBSTR (NAMES (J-1), K, 1)
   31                                    THEN DO;        K=K+1;        GO TO TEST;
   34                                END;
   35                               IF SUBSTR (NAMES (J), K, 1) = '.' |
                                    SUBSTR (NAMES (J), K, 1) < SUBSTR (NAMES (J-1), K, 1
   36                                    THEN CALL SWITCH;
   37                          IF J< NN-CT  THEN GO TO SCAN;
   39                     IF INDICATOR='1'B THEN GO TO SORT;
   41                END ALPHA;
   /* THE INTERNAL PROCEDURE SWITCH IS USED BY BOTH ALPHA AND NUMERIC TO REVERSE THE ORDER
         OF THE SCORES AND THE NAMES WHILE SORTING THEM.      */
   43                SWITCH: PROCEDURE;
   44                     INDICATOR='1'B;
   45                     SCORE=SCORES (J);              NAME=NAMES (J);
   47                     SCORES (J)=SCORES (J-1);       NAMES (J)=NAMES (J-1);
   49                     SCORES (J-1)=SCORE;            NAMES (J-1)=NAME;
   51                RETURN;
   52                END SWITCH;
   53           RETURN;
   54           END SORT;
```

лиц (числа разделены пропусками). Программа считывает эти данные, распределяет номера счетов в порядке убывания и определяет средний номер счета. Затем программа выдает на печать результаты, располагая номера счетов в нисходящем порядке. За этой таблицей следует таблица с алфавитным списком лиц и их счетов. В конце дается среднее значение номеров счетов.

В программу включено несколько примечаний, чтобы подчеркнуть некоторые ее особенности Если читатель уже знаком хотя бы с одним языком высокого уровня, то ему будет сравнительно легко ознакомиться с программой. Следует отметить несколько особенностей языка ПЛ/1 в этой программе: внутренние и внешние процедуры подпрограмм, удобная обработка строки символов, легкое управление памятью, «переменная размерность», использование двух различных типов операторов ввода-вывода, расширенное множество символов, функция массива SUM в строке 15 процедуры E201 и расширенные возможности операторов DO и IF.

Комментарии к программам

Программа E201

Перед оператором 1:

/*Глава 2, пример 1*/
/*Сортировка числовая и алфавитная*/

После оператора 1:

/*Использование данных CONTROLLED в операторе DECLARE (сокращенно DCL) позволяет программисту распределять память во время выполнения программы. Внешний ATTRIBUTE минимизирует необходимость прохождения данных в процедуры через список аргументов. Этот оператор похож на оператор COMMON в Фортране и детально будет описан в главе 6.*/

После оператора 3:

/*Оператор ALLOCATE определяет требования к памяти для хранения данных CONTROLLED во время выполнения программы.*/

После оператора 4:

/* Сначала считать имена и номера счетов, далее просканировать их и найти максимальный и минимальный, затем вычислить средний.*/

Перед оператором 17:

/*Заметьте, что команда DO LOOP в операторе 17 отличается по форме от команд в операторах 5, 7 и 11. ПЛ/1 позволяет использовать оба варианта с несколькими расширениями, что очень ценно для программиста.*/

После оператора 17:

/* Оператор обращения образует безусловный переход к другой процедуре (здесь внешней): переменная DATA TYPE, объявлена строкой символов длины 6, передается процедуре SORT (сортировка).*/

Программа SORT

После оператора 1:

Использование '*' в описании размерности определяет, что размерность переменного массива такая же, как в последнем операторе ALLOCATE.

После оператора 6:

Внутренняя процедура NUMERIC сортирует счета в порядке убывания.

После оператора 20:

Внутренняя процедура ALPHA сортирует имена в алфавитном порядке.

После оператора 41:

Как процедура ALPHA, так и процедура NUMERIC используют внутреннюю процедуру SWITCH для реверсирования порядка счетов и имен во время сортировки.

ТАБЛИЦА СЧЕТОВ В ПОРЯДКЕ УБЫВАНИЯ

Имя	Счет
DOE, RALPH M.	61
SMITH, ROBERT R.	59
MILLER, ALEX H.	54
SMITH, RICHARD J.	52
JONES, RALPH D.	51

ТАБЛИЦА СЧЕТОВ В АЛФАВИТНОМ ПОРЯДКЕ

Имя	Счет
DOE, RALPH M.	61
JONES, RALPH D.	51
MILLER, ALEX H.	54
SMITH, RICHARD J.	52
SMITH, ROBERT R.	59

THE AVERAGE SCORE IS 55

2.2. НАБОРЫ СИМВОЛОВ

Любой язык состоит из элементов, организованных определенным образом. Основные элементы языка — символы, которыми этот язык записывается. В ПЛ/1 существуют два основных набора символов: набор из 60 символов и набор из 48 символов. Теоретически выбор одного из этих двух наборов произволен; практически же этот выбор зависит от устройства ввода-вывода, применяемого в системе. В каждом наборе существуют три вида символов: *алфавитные*, *цифровые и специальные*. Если алфавитные и цифровые символы объединяются, их называют *алфавитно-цифровыми*. Некоторые символы могут быть объединены, они образуют *комбинированные* символы.

В настоящей книге мы будем пользоваться набором из 60 символов. Всюду, где устройства ввода-вывода позволяют, рекомендуется именно этот набор. Он дает возможность составить наиболее точные, легко читаемые программы и сократить до минимума количество ошибок.

Далее перечислены символы этого набора:

28 алфавитных символов:
26 букв английского алфавита
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
знак доллара $
знак номера #
10 цифр:
0 1 2 3 4 5 6 7 8 9
21 специальный символ:
пробел
знак равенства =
плюс +
минус −
звездочка (знак умножения) *
наклонная черта (знак деления) /
открывающая скобка (
закрывающая скобка)
запятая ,
точка .
апостроф (кавычка) '
процент %
точка с запятой ;
двоеточие :
символ отрицания ¬
символ И &
символ ИЛИ |
знак больше >
знак меньше <
символ разбивки (черта под строкой) _
вопросительный знак ?

(В настоящее время символ «вопросительный знак» в языке не имеет специального значения).

Общая таблица наборов из 60 и 48 символов, комбинированные символы, коды для перфорирования и 8-битовый код EBCDIC (Extended Binary Coded Decimal Interchange Code) даны в приложении А.

2.3. ОСНОВНОЙ СИНТАКСИС ПЛ/1

Правильная организация элементов внутри языка называется синтаксисом*. В языке ПЛ/1 программа составляется из выражений и основных операторов, которые могут быть либо простыми, либо комбинированными. Операторы объединяются в программные сегменты, называемые блоками процедур и блоками начала. Такая организация похожа на ранее применяемые языки, такие, как Фортран, где основными единицами служили выражения и операторы Фортрана, а они, в свою очередь, могли образовывать основную программу или подпрограммы.

Блоки процедур ПЛ/1 могут быть составлены отдельно, а затем соединены вместе, если это необходимо. Точно такая же возможность имеется в Фортране, когда отдельно составляется основная программа и подпрограммы, а затем из них составляется единая программа. Однако в отличие от Фортрана ПЛ/1 позволяет пересечение блоков, и обмен результатами между блоками простой и прямой. В каждой программе на ПЛ/1 должна быть процедура, которую программа определяет как *основную*. Программа может быть сложной, в основную процедуру могут быть вставлены блоки; другие блоки могут добавляться или составляться отдельно. Выполнение программы начинается с основной процедуры. Мы ограничимся сейчас написанием программ с основной процедурой, в которой нет вставных блоков. Хотя это довольно существенное ограничение, но возможности ПЛ/1 позволяют решать многие задачи даже при такой простой структуре программ.

Формат перфокарт. Предполагается, что в ПЛ/1 ввод и вывод будут представлять собой непрерывный поток информации. Это вполне возможно при современных средствах обработки данных на расстоянии. Применительно к перфокарте это означает, что колонки 2—72 читаются с первой карты колоды до последней так же, как при нанесении информации на одну данную полосу перфоленты. Следовательно, карта не является основной единицей информации. На ней может быть отперфорирован один оператор или даже часть его, может быть отперфорировано также много таких операторов.

Колонку 1 перфокарты оставляют пустой, а колонки 73—80 обычно не сканируются транслятором.

Использование колонок перфокарт для операторов. Колонка 1 должна быть оставлена пустой, колонки 2—72 — исходные операторы, колонки 73—80 не сканируются транслятором, программист может пользоваться ими для нумерации или специальной идентификации.

Конец оператора ПЛ/1 обозначается точкой с запятой. *Каждый оператор ПЛ/1 должен заканчиваться точкой с запятой* (;). За некоторым исключением пропуски в языке не учитываются, с их помощью программу делают более удобной для чтения. Пропуски могут быть в начале и конце любого оператора, перед операторами или после них, а также после большинства других ограничителей.

---

* Обычно синтаксисом языка называется множество формальных правил вывода правильно построенных строк или предложений языка. — *Примеч. пер.*

Коммеитарии. Коммеитарии можио вставить в любой пропуск. Они ие принимаются во внимаиие трансляторoм, а используются для документации.

Обычная форма коммеитария может быть такой:

/*Любая строка символов ПЛ/1*/

Символы /* означают иачало коммеитария, а символы*/ — его коиец. *Между двумя этими символами не может быть никакого пропуска.*

Пример

```
/* ЭТОТ ОПЕРАТОР МОЖНО ВСТАВИТЬ В ЛЮБОЕ МЕСТО ПРОГРАММЫ НА ПЛ/I,
                    ГДЕ ЕСТЬ ПРОПУСК*/
```

Неправильно набитые карты, на которых отперфорированы комментарии, могут дать некоторые необычные результаты. Например, если два зиака — */, которые стоят в конце комментария, отперфорироваиы на карте после колоики 72, машина ие воспринимает их, и, следовательно, следующие операторы ПЛ/1 будут интерпретироваиы как комментарии. С другой стороны, в некоторых системах символы /* могут служить ограничителями, если они иаиесены в колоиках 1 и 2. В этом случае, так же как и в операционной системе-360, коммеитарий, который начинается в колонке 1, вызовет прекращеиие чтения программы. Другими словами, необходимо твердо следовать правилам перфорирования данных на карты так, как они даны в настоящей книге.

Идентификаторы. Имена на ПЛ/1 задаются примерно так же, как на ранее известных языках, таких, как Фортран и Кобол. В ПЛ/1 эти имена иезависимо от того, для чего оии испольэуются, называются *идентификаторами*. Идентификатор может содержать от одиого буквенного символа до целой строки буквенио-цифровых символов и символа — . Перед идентификатором или после иего может стоять пропуск или какой-либо другой ограиичитель. Если используется более чем одии символ, то первый должен быть буквеиным. Длииа строки идентификатора не может превышать 31 символа. (Существует иесколько исключений из этого правила, об этом будет сказано, когда они встретятся.)

Примеры

```
A
$XYZ
A21
RATE___PER___HOUR
NAME___ADDRESS
 # 789
```

Метки. Первое, что может заметить программист, знакомый с Фортраном и Коболом, когда ои иачииает изучать программу иа ПЛ/1, — это отсутствие номеров операторов или строк. Вместо иих в ПЛ/1 используются идентификаторы. В этом случае оии определяются как *метки*. Оператор ПЛ/1 может иметь более чем одну метку, и любая из них может быть отнесеиа к даиному оператору. Метка должиа стоять перед оператором и отделяться от оператора двоеточием (:). Метка ставится перед оператором, только если в этом есть иеобходимость.

Форма простого оператора ПЛ/1 выглядит следующим образом:

label: label: ... label: оператор PL/1;

некоторые или все метки могут быть опущены.

PROCEDURE и END. Как указывалось ранее, в каждой программе ПЛ/1 должна быть главная процедура. В ней могут также быть, хотя и необязательно, другие блоки. Первый оператор в этой главной процедуре должен быть записан следующим образом:

метка: PROCEDURE OPTIONS (MAIN);

Первый оператор должен иметь одну и только одну метку. Эта метка является идентификатором, но с одним исключением: ее длина не должна превышать семи символов[1].

Для того чтобы отметить конец блока этой процедуры, нужно указать оператор END в форме:

label: label: ... label: END label;

Это последний оператор в процедуре. Метки в этом операторе END произвольны. Первая метка (или метки) является метками оператора, в то время как последняя метка, если она необходима, означает метку процедуры. Это облегчает чтение блока процедуры.

Пример

Если первый оператор процедуры

ABC: PROCEDURE OPTIONS (MAIN);

то любой следующий оператор может быть последним:

```
END;
END ABC;
XYZ: END;
XYZ: END ABC;
```

Если во время выполнения программы встречается оператор END, то выполнение процедуры прекращается. Оператор END в ПЛ/1 может использоваться и в других случаях. Об этом будет сказано далее.

Ключевые слова. В некоторых ранее созданных языках ключевые слова, вставленные в программу, приносили много трудностей. Если ими пользовались, не зная их истинного назначения, то это, как правило, приводило к получению неправильных результатов, которые с трудом поддавались определению. На практике это означало, что программист должен был либо знать заранее все ключевые слова и их употребление, либо идти на риск неправильного выполнения программы. В ПЛ/1 ключевых слов гораздо больше, чем в Фортране, но программисту нет необходимости знать о них, если только он не хочет использовать эти слова в их формальном значении. Транслятор переведет ключевое слово в контексте оператора, если оно там встретится. Например, ключевое слово в Фортране SIN может быть взято в ПЛ/1 в качестве имени метки или переменной, хотя в ПЛ/1 имеется встроенная функция с таким названием. Намерен ли программист словом

[1] В некоторых вариантах трансляторов для этой метки достаточно только шести символов.

SIN обозначить функцию или метку, становится ясным из структуры программного оператора ПЛ/1. Список ключевых слов и их функций дан в приложении В. Ключевые слова в приведенных примерах будут определяться по мере их введения.

2.4. КОНСТАНТЫ, ПЕРЕМЕННЫЕ И ОПЕРАТОР DECLARE (ОБЪЯВИТЬ)

Данные. В одной из версий Фортрана шесть типов данных: целые, с плавающей запятой, двойной точности, комплексные, логические и символьные. Это затрудняет и запутывает дело. ПЛ/1 позволяет использовать все эти типы данных и даже больше, но употребление их в ПЛ/1 гораздо проще.

Для определения типа переменных и констант существуют следующие описатели:

Основание системы счисления ... может быть BINARY (двоичное) или DECIMAL (десятичное)
Масштаб ... может быть FIXED (с фиксированной точкой) или
FLOAT (с плавающей точкой)
Тип ... может быть REAL (действительный) или COMPLEX (комплексный)
Точность .... может быть определена программистом в пределах применяемого транслятора

В подмножестве комплексный тип не используется. Предполагается, что все арифметические данные вещественные и описание REAL нельзя употреблять в операторе DECLARE.)

Перед всеми арифметическими константами может стоять знак плюс или минус. Если такого знака нет, то эти константы полагаются положительными. Каждая арифметическая константа или переменная принадлежит одному из перечисленных типов*. В электронной вычислительной машине различные данные представляются разными способами. Например, число 27 в двоичной форме записывается как 11011 и означает $1(2)^4 + 1(2)^3 + 0\,(2)^2 + 1(2)^1 + 1(2)^0$. Та же самая величина может быть представлена как 11110010 11110111. В последнем случае каждое десятичное число представлено в коде, пригодном для многих современных вычислительных машин. В приложении А дан такой код для всех символов ПЛ/1.

DECIMAL FIXED POINT (Десятичные константы с фиксированной точкой). Десятичные константы с фиксированной точкой представляют собой строку из одной или нескольких десятичных цифр с десятичной точкой или без нее. Если десятичной точки нет, то предполагается, что она следует сразу же за последним числом строки.

Примеры

```
8 7625
248
.00007
972
+10.2
```

---

* Г. е. является двоичной или десятичной, фиксированной или с плавающей точкой, действительной или комплексной. — *Примеч. пер.*

Зиак плюс разъясияет программисту порядок использоваиия константы.

Для того чтобы определить перемениую ABC со свойствами REAL, DECIMAL, FIXED (вещественная, десятичная, с фиксированиой точкой) пятью (5) цифрами с плюсом или мииусом, из которых три идут после десятичной точки, можио воспользоваться следующим оператором DECLARE:

```
DECLARE ABC RFAL DECIMAL FIXED (5.3);
```

(В подмножестве языка оператор DECLARE следует сразу же за оператором PROCEDURE.)

Пропуски нужны для разделения различиых слов в операторе. Порядок, в котором перечислены свойства оператора DECLARE, произвольный. Если в скобки заключено только одио число, например (5), то точность перемеиной определена, и она представляет целое число. Эта запись эквивалеитна описанию точиости (5, 0).

Примеры

Определение точности	Цифровые значения (знак не включен)
(5, 3)	12.345
(5)	12345
(7)	1234567
(7, 2)	12345.67
(8, 4)	1234.5678

Двоичные констаиты с фиксированной точкой. Двоичные константы с фиксированной точкой представляют собой строку с одним или иескольким двоичиыми числами с двоичной точкой или без иее. В коице стоит буква B.

Примеры

```
1101100 B
11.011  B
—100.10B
10.     B
+1      B
```

Способ описаиия действительной переменной NUM с двоичной фиксированной точкой и с заданиой точностью аналогичен способу, применяемому в случае десятичной фиксированной точки. Например,

```
DECLARE NUM REAL BINARY FIXED (12.3);
```

объявляет, что перемеииая NUM представляет собой перемеиную с двоичной фиксированиой точкой, содержащей 12 двоичных чисел, три из которых следуют за двоичной точкой. Двоичиое число с такой точиостью равио

```
—101101000.001
```

(В подмиожестве языка коистаита с двоичной фиксированиой точкой не может содержать эту точку.)

Десятичные константы с плавающей точкой. Употребление десятичиых констант с плавающей точкой осиовано на тех же принципах, что и употребление чисел с плавающей точкой в Фортраие. Место десятичной точки определяется числом с буквой E. Например, E5 означает, что

десятичная точка будет сдвинута на пять знаков вправо, а E — 5 означает, что десятичная точка будет сдвинута на пять знаков влево.

Примеры

Десятичные числа с плаающей точкой	Числоаые значения
20.E5	2000000.
11E2	1100.
20.6E—10	.00000000206
—.865421E14	—8654210000000.

Точность определяется десятичным целым числом, которое показывает, сколько знаков должно предшествовать букве E. Оператор

```
DECLARE XYZ REAL DECIMAL FLOAT (4);
```

объявляет, что переменная XYZ представляет собой константу с плавающей десятичной точкой; перед экспонентой E стоит четыре десятичных числа.

Двоичные константы с плавающей точкой Двоичные константы с плавающей точкой состоят из поля двоичных чисел, которое называется мантиссой и обозначается буквой E, и экспоненты, которая выражается десятичными числами и заканчивается буквой B. Перед мантиссой может стоять знак, но его может и не быть. Двоичная точка в мантиссе необязательна.

Примеры

Двоичные числа с плавающей точкой	Даоичные цифровые значения
1101.1E4B	11011000
—1001.111E—2B	—10.01111
1101E24B	11010000000000000000000000

Десятичная экспонента показывает место расположения двоичной точки. Точность определяется так же, как в случае десятичного числа с плавающей точкой.
Оператор

```
DECLARE BCD REAL BINARY FLOAT (10);
```

объявляет, что переменная BC есть действительная, двоичная переменная с плавающей точкой; перед E стоит 10 двоичных цифр.

Отбрасывание значащих цифр. В большинстве электронных вычислительных машин десятичные и двоичные константы с плавающей точкой хранятся в машине, при этом подразумевается, что десятичная точка стоит перед первой цифрой, не являющейся нулем. Если константа присваивается переменной с плавающей точкой и объявленная точность переменной меньше точности константы, то отбрасываются цифры младших разрядов. Если константа присваивается переменной с фиксированной точкой , а объявленная точность меньше, чем точность константы, то старшие или младшие разряды отбрасываются в зависимости от описания точности переменной.

Примеры
Переменная с плавающей точкой: значение константы 12345.678

Описание точности	Значение, приписываемое переменной
(6)	.123456E5
(8)	.12345678E5
(10)	.1234567800E5

Переменная с фиксированной точкой: значение константы 12345.678

Описание точности	Значение, приписываемое переменной
(8,3)	12345.678
(6,3)	345.678
(8,4)	2345.6780
(9,2)	0012345.67
(8)	00012345.

В случае переменной с фиксированной точкой корректировка всегда производится по десятичной точке, а поле либо сокращается, либо к нему добавляются нули справа и слева от десятичной точки.

Комплексные переменные. В подмножестве языка не выполняются операции с мнимыми и комплексными числами. Мнимая константа может быть записана как любой из предыдущих типов данных, за ней сразу же следует буква I.

Примеры

Мнимая константа	Значение
2I	2I
—2.34I	—2.34I
2.34E2I	234I
101.01BI	101.011 (двоичное число)
110I 011E—2BI	11.0101II (двоичное число)

Комплексная константа, так же как в математике, записывается как сумма вещественной и мнимой константы.

Примеры

Комплексная константа	Десятичное число
3—4I	3—4I
4.56+2.34I	4.56+2.34I
—1B+11BI	—1+3I

Комплексная переменная может быть тех же типов, что и вещественная переменная. Эти признаки относятся как к вещественной, так и к мнимой части комплексного числа. Оператор

```
DECLARE Z COMPLEX FIXED DECIMAL (5,2);
```

объявляет, что переменная Z — комплексная переменная с фиксированными вещественной и мнимой частями, выраженная десятичными числами точности (5,2).

Начальные значения. Начальные значения для переменных могут быть определены в операторе DECLARE записью INITIAL (значение). Оператор

```
DECLARE A REAL FIXED DECIMAL (5,0) INITIAL (0), B REAL DECIMAL
FLOAT (6) INITIAL (2E1);
```

присваивает значение 0 вещественной фиксированной десятичной переменной A точности (5,0), а значение 20 — вещественной десятичной переменной с плавающей точкой точности (6). В операторе INITIAL

может быть использован повторитель как средство инициализации значений массива. Например,

```
DECLARE A (100) REAL FIXED DECIMAL INITIAL ((10)1, (90)0);
```

будет означать A (1), ..., A (10) со значением 1, а остальные 90 — со значением 0. Описатель типа может относиться к нескольким переменным, если они одного и того же типа. Например, оператор

```
DECLARE A FIXED REAL DECIMAL (5,0), B FIXED REAL BINARY (10,2);
```

эквивалентен

```
DECLARE (A DECIMAL (5,0), B BINARY (10,2)) FIXED REAL;
```

Описатели типа могут идти в любом порядке за одним исключением: описатель точности не может следовать непосредственно за именем переменной. Это объясняется тем, что оператор DECLARE служит также для определения размерности переменной, как оператор DIMENSION (размерность) в Фортране, и эта информация о размерности следует сразу же за именем переменной. Оператор

```
DECLARE (A(10), B (8, 10)) REAL DECIMAL FLOAT (7);
```

объявляет, что A есть переменная размерности 10, а B — массив 8×10. Все эти 90 элементов представляют собой вещественные десятичные переменные с плавающей точкой точности 7. Вопросы, связанные с размерностью, подробнее будут рассмотрены в следующей главе.

Язык ПЛ/1 представляет прекрасные возможности для индексирования. Для одной переменной может употребляться до 32 индексов. Эти индексы бывают положительными, отрицательными или нулевыми. Сами индексы могут быть выражениями ПЛ/1. (Подмножество языка позволяет использовать максимум три индекса.) На практике нет необходимости проводить спецификацию всех определителей переменных. Если какие-либо определители не обозначены, то система будет определять их по умолчанию. Правила для определения переменных по умолчанию изложены в следующей главе. Большинство программистов, работающих с Фортраном, при изучении ПЛ/1 слишком полагаются на эти правила по умолчанию. До приобретения опыта по программированию на ПЛ/1 лучше включить все переменные в оператор DECLARE.

2.5. ВЫРАЖЕНИЯ И ОПЕРАТОР ПРИСВАИВАНИЯ

Арифметические операции на языке ПЛ/1 аналогичны операциям на Фортране.

Оператор	Действия
+	плюс
−	минус
*	умножение
/	деление
**	возведение в степень

Выражения записываются и выполняются так же, как на Фортране, за одним исключением: делается различие между знаками плюс и ми-

нус, стоящими перед операндом или выражением и этими же знаками, соединяющими два операнда. В первом случае знак плюс или знак минус называется префикс, во втором — инфикс. В последовательности символов — $A + B$ знак минус является префиксом, а знак плюс — инфиксом. При возведении в степень операторы с префиксом имеют самый высокий приоритет при выполненни операцнй. Порядок выполнения арифметических операций в ПЛ/1 следующий:

```
**, префикс +, префикс — самый высокий уровень
*,/                    ↓
инфикс +, инфикс —  самый низкий уровень
```

Как и в Фортране, операции с одннаковым приоритетом выполняются в выражении слева направо. К исключению относятся три оператора самого высокого приоритета, которые выполняются справа налево. Выражение — A + B* — C**2 обрабатывается как (—A) + (B*(— — (C**2))). Прн написании программы на ПЛ/1 круглые скобки необходимы для того, чтобы операции правильно выполнялись.

Оператор присваивания A = B + C; имеет тот же смысл, что и в Фортране, т. е. переменной A присваивается значение B+C. Этот оператор может прнвести к изменению типов данных. Выражение слева от знака равенства в операторе присваиванння может иметь несколько переменных н всем им может быть прнсвоено значение выражения, стоящего справа от знака равенства. Например, в выраженни X, Y, Z = A + B; значение A + B присваивается X, Y и Z. (В подмножестве языка не допускается присваивание одного значения нескольким переменным. Приведенный пример должен быть записан как три отдельных оператора:

$$X = A + B;\ Y = A + B;\ Z = A + B;$$

или

$$X = A + B;\ Y = X;\qquad Z = X;$$

последний вариант дает более эффективную рабочую программу.) В ПЛ/1 гораздо больше типов данных, чем в Фортране и Коболе, и программист, если это целесообразно, может употреблять различные типы в одном выражении. Правила обработки выражений, содержащих смешанные типы данных, довольно сложны. Выражения записываются таким образом, чтобы до тех пор, пока это не является абсолютно необходимым, они не преобразовывались. Это делается с целью сохранения, насколько возможно, точности и значения промежуточных и конечных результатов.

Преобразование арифметических данных в выражении производится следующим образом. С каждой операцией связан один или два операнда, которые могут быть константой, переменной или ранее вычисленным выражением. Если каждый операнд является константой, то преобразование происходит во время трансляции. Если один операнд представляет собой константу с плавающей точкой, а другой— с фиксированной точкой, то операнд с фиксированной точкой может

быть преобразован в операнд с плавающей точкой. Если один операнд — двоичное число, а другой — десятичное, то десятичное число преобразовывается в двоичное. Если один операнд комплексный, то результат тоже будет комплексным, а вещественный операнд не преобразуется.

В начале обучения студенту лучше поступать так, как этого требует здравый смысл, и не смешивать слишком много типов данных в одном операторе. Более подробно правила преобразования арифметических данных, а также другие виды преобразований будут обсуждаться в главе 4.

Для того чтобы проиллюстрировать последовательность операций и преобразования данных, рассмотрим следующий пример:

```
Z = (X**2 - 3*Y + (1 + 2)*X)*C;
```

Будем считать, что в этом примере Z и C объявлены комплексными, X и Y — действительными и все четыре числа являются десятичными с плавающей точкой. Кроме того, число 1 объявлено десятичным, действительным, с плавающей точкой. Это выражение будет вычисляться в следующей последовательности:

Порядковый номер	Операция	Действие
1	1 + 2	Преобразование не производится. В результате получается десятичное вещественное число с плавающей точкой
2	X**2	Преобразование не производится — вычисляется как X*X
3	3*Y	3 преобразуется к виду с плавающей точкой во время трансляции. В результате получается число с плавающей точкой
4	(1 + 2)*X	(1 + 2) преобразуется к виду с плавающей точкой. В результате получается число с плавающей точкой
5	(X**2) — (3*Y)	Преобразование не производится. В результате получается число с плавающей точкой
6	(X**2 — —3*Y) + ((1 + 2)*X)	Преобразование не производится. В результате получается число с плавающей точкой
7	((X*2 — 3*Y) + +((1 + 2)*X))*C	Преобразование не производится. В результате получается комплексное число.
8. Полученное значение комплексного числа присваивается переменной Z.		

Для обработки большего набора типов данных ПЛ/1 располагает гораздо большим числом встроенных функций, чем Фортран. Список этих функций приводится в приложении Д. В большинстве случаев основание, масштаб и тип аргумента математической функции определяют результат. Например, функция SIN используется для вычисления синуса вещественного или комплексного аргумента с фиксированной или плавающей точкой, десятичного или двоичного. Наиболее часто встречающимися математическими функциями являются:

Функция (аргумент)	Получающийся результат
EXP (X)	e^x
LOG (X)	$\log_e x$
SIN (X)	синус x
COS (X)	косинус x
TAN (X)	тангенс x
SQRT (X)	квадратный корень из числа x
ABS (X)	абсолютное значение числа x
MOD (X, Y)	остаток от деления целых чисел x/y
FLOOR (X)	наибольшее целое число, не превосходящее x
CEIL (X)	наименьшее целое, превосходящее x

2.6. ОПЕРАТОРЫ УПРАВЛЕНИЯ

GO TO [ПЕРЕЙТИ НА]. В ПЛ/1 имеется только один оператор GO TO. Он имеет форму

```
GO TO метка;
```

Как всякий оператор языка ПЛ/1, оператор GO TO может иметь одну или много меток. Приведем примеры правильных операторов GO TO:

```
    GO TO START;
ABC: GO TO OUTPUT;
X:Y:  GO TO CHANGE__DUE;
```

Может показаться, что эти операторы выполняются так же, как безусловный оператор GO TO п в Фортране, но позже будет показано, что это не так; в ПЛ/1 метки могут быть переменными и могут даже индексироваться. Это, безусловно, увеличивает возможности оператора GO TO в ПЛ/1 по сравнению с Фортраном.

DO [ВЫПОЛНИТЬ]. В ПЛ/1 оператор DO имеет гораздо большие возможности, чем этот же оператор в Фортране. Прежде всего, представим оператор цикла Фортрана в его наиболее общей форме:

```
n1 DO  n2  I = m1, m2, m3
           :
           :
```

n_2 последний оператор в цикле.

В ПЛ/1 этот цикл может быть записан следующим образом:

```
метка 1: DO I = m1  TO m2 BY m3;
                    :
                    :
метка 2: END метка 1;
```

Метка 1 и метка 2 произвольные. Если в операторе DO метка опущена, то ее нельзя использовать после оператора END. Как и в Фортране, если BY опущено, то значение m_3 принимается за 1. Последним оператором в цикле должен быть оператор END. m_1, m_2 и m_3 могут быть выражениями, а не только целыми числами. Приведем несколько примеров этой формы оператора цикла DO:

```
A:  DO I=1 TO 50 BY 2;
           :
           :
END;
```

Предложения в BY и TO могут меняться местами, как это видно из следующего примера:

```
START: DO I=1 BY 2 TO 100;
       DO J=I-1 TO 200;
                 .
                 .
                 .
   B:  END;
       END;
```

В этом примере операторы DO вложены друг в друга, причем каждому оператору DO соответствует собственный оператор END:

```
DO X=1 TO 5, 10 TO 8 BY-. 1, 0, 3.14159;
                 .
                 .
                 .
END;
```

В этом примере цикл будет выполняться при X, равном от 1 до 5 с приращением 1, затем при X, равном от 10 до 8 с приращением — 1, затем один раз при X, равном 0, и, наконец, еще один раз при X, равном 3.14159.

Следующее выражение неправильно:

```
DO I=1 TO 5, J=1 TO 10;
          .
          .
          .
END;
```

Это выражение должно записываться в виде двух отдельных циклов DO, и каждый цикл должен иметь собственный оператор END, так как в операторе DO в качестве индекса можно использовать только одну переменную. В операторе DO могут быть определены особые значения индекса, как это сделано в следующем примере:

```
DO X=1, 8,3.4, N, VALUE;
          .
          .
          .
END;
```

Правила входа и выхода из операторов циклов DO и правила вычисления циклов аналогичны правилам, принятым в Фортране. Однако необходимо заметить, что правило окончания каждого цикла оператором END позволяет обойти запрещение Фортрана использовать операторы передачи управления в качестве последних операторов цикла. Это делает программирование на ПЛ/1 более естественным, а программу — легко читаемой.

Конструкция WHILE. Существует одна дополнительная конструкция, которая может быть включена в оператор DO, а именно конструкция WHILE.

Следующий пример иллюстрирует конструкцию WHILE:

```
A:  DO WHILE (X < Y);
          .
          .
          .
END A;
```

Этот цикл будет выполняться один раз, а затем текущие значения X и Y будут сравниваться. Если значение выражения $X < Y$ истинно, то цикл выполняется повторно. Цикл заканчивается, если значение выражения $X < Y$ становится ложным (т. е. $X > Y$) при про-

верке в конце одного из прохождений цикла. Необходимо заметить, что если X становится больше или равным Y в середине цикла, то вычисления продолжаются. Проверка конструкции WHILE производится только в конце цикла.

```
DO H=10 TO 100 BY 5 WHILE (X < Y);
                  :
END;
```

Этот цикл будет выполняться при H, первоначально равном 10, затем с приращением 5 до тех пор, пока при проверке перед очередным прохождением цикла не будет выполнено одно из двух условий. Цикл будет выполняться повторно, если $H \leqslant 100$ и если текущее значение X и Y удовлетворяет отношению $X < Y$. Если ни одно из этих условий не выполняется, то цикл продолжает повторяться.

В ранее приведенном примере выражение в скобках, следующее за конструкцией WHILE, может быть гораздо более сложным, чем это показано. Прежде чем проиллюстрировать это утверждение, мы должны кратко обсудить операторы сравнения и логические операторы. Они во многом аналогичны операторам Фортрана, но из-за расширенного набора символов обозначения для операторов ПЛ/1 отличны от Фортрана.

В операциях сравнения пользуются следующими символами:

равно	=
не равно	¬=
больше чем	>
не больше чем	¬>
больше или равно	>=
меньше чем	<
не меньше чем	¬<
меньше или равно	<=

В логических операциях используются символы:

не	¬
или	|
и	&

Ранее рассмотренные операторы, логические и арифметические операции и операции сравнения выполняются в следующей последовательности:

**, ¬ (логический), префикс +, префикс — самый высокий приоритет
/, * второй приоритет
инфикс +, инфикс — третий приоритет.

Все операции сравнения:

=, ¬=, >, ¬>, >=, <, ¬<, <= четвертый приоритет
& (логический) пятый приоритет
| (логический) шестой приоритет.

Когда в одном и том же выражении встречаются два или более оператора самого высокого приоритета, то порядок их выполнения справа налево. Когда в одном выражении встречается два или более

оператора одного и того же приоритета, кроме наивысшего, то очередность их выполнения — слева направо. Круглыми скобками можно пользоваться для изменения приоритета, так как выполнение оператора начинается с внутренних круглых скобок.

Основное различие между ПЛ/1 и ранее созданным Фортраном состоит в том, что в ПЛ/1 операторы префикс + и инфикс — выполняют в порядке самого высокого приоритета. Анализ логических операторов в ПЛ/1 выполняется проверкой строк битов. Позже такая проверка будет рассмотрена детально. А сейчас рассмотрим логическое выражение как выражение, определяющее значение TRUE (истинно) или FALSE (ложно) при выполнении программы с учетом очередности приоритетов и следующей таблицы истинности:

Выражение	¬ (Выражение)
T	F
F	T

Выражение 1	Выражение 2	(Выражение 1) & (Выражение 2)	(Выражение 1) | (Выражение 2)
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F

Например, логическое выражение

$$XX^{**} - 2 < A*B + C \,\&\, A = B$$

будет выполняться, как если бы оно было записано в виде

$$((X^{**}(-2)) < ((A*B) + C)) \,\&\, (A = B)$$

Возвращаясь к нашему обсуждению конструкции WHILE в операторе DO, можно сказать, что выполнение цикла DO будет повторяться до тех пор, пока логическое выражение в конструкции WHILE будет истинно и индекс цикла, если только он имеется, не превзойдет своего конечного значения. Необходимо следить за тем, чтобы цикл мог закончиться в какой-либо точке.

Например, цикл

```
DO WHILE (A-1=B);
        .
        .
        .
END;
```

где A — вещественное десятичное число, а B — вещественное двоичное число, может привести к тому, что цикл будет повторяться бесконечно, так как A никогда не может быть в действительности равно B из-за разницы во внутреннем представлении чисел.

Группа DO. Существует еще одна форма оператора DO, которая относится только к последовательности операторов, образующих единую группу. Вот пример такой группы:

```
метка 1: DO;
        .
        .
        .
метка 2: END метка 1;
```

Как и раньше, метки необязательны.

Операторы в группе DO выполняются только один раз. Использование группы DO станет понятным после рассмотрения оператора IF.

Оператор IF. В Фортране существуют два оператора IF: арифметическое IF и логическое IF. В ПЛ/1 имеется только один оператор IF, но его использование значительно расширено.

Оператор IF THEN. Основная форма такого оператора следующая:

необязательная метка (метки): IF логическое выражение THEN необязательная метка (метки): один оператор ПЛ/1 или оператор DO;

Этот оператор выполняется аналогично логическому оператору IF в Фортране. Это значит, что если логическое выражение истинно, то сначала выполняется оператор, следующий за THEN, или оператор DO, а затем следующий оператор ПЛ/1 в порядке очередности, если в конструкции THEN не будет осуществлена передача управления на другой оператор. Если логическое выражение ложно, то конструкция THEN пропускается, а выполняется следующий за конструкцией THEN оператор. Эту операцию иллюстрирует блок-схема, представленная на рис. 2.1.

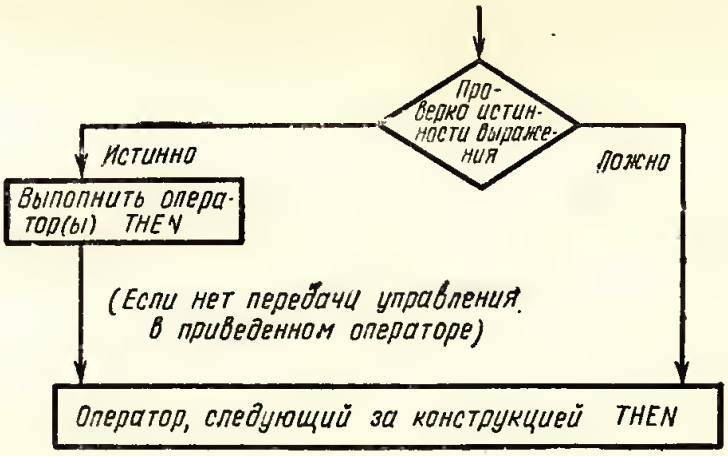


Рис. 2.1.

Пример

Фортран	ПЛ/1
IF (A. LT. B) X=A+B	IF A < B THEN X=A+B;
Y=3.* X	Y=3*X;
IF (A. GT. B) GO TO 35	IF A <=B THEN DO;
X=A+B	X=A+B;
C=D+F	C=D+E;
35 F=G+H	EHD;
	F=G+H;

Оператор IF — THEN—ELSE. Общая форма оператора имеет вид:

необязательная метка (метки): IF логическое выражение THEN необязательная метка (метки):

один оператор ПЛ/1 или оператор DO

ELSE необязательная метка (метки):

один оператор ПЛ/1 или оператор DO;

Если логическое выражение истинно, то выполняется конструкция THEN (которая может состоять из одного оператора или группы операторов), а вся конструкция ELSE пропускается. Затем будет выполняться первый оператор, непосредственно следующий за конструкцией ELSE, если в конструкции THEN не было передачи управления на другой оператор. Если логическое выражение ложно, то вся конструкция THEN пропускается, а выполняется конструкция ELSE. Эту операцию иллюстрирует блок-схема, представленная на на рис. 2.2.

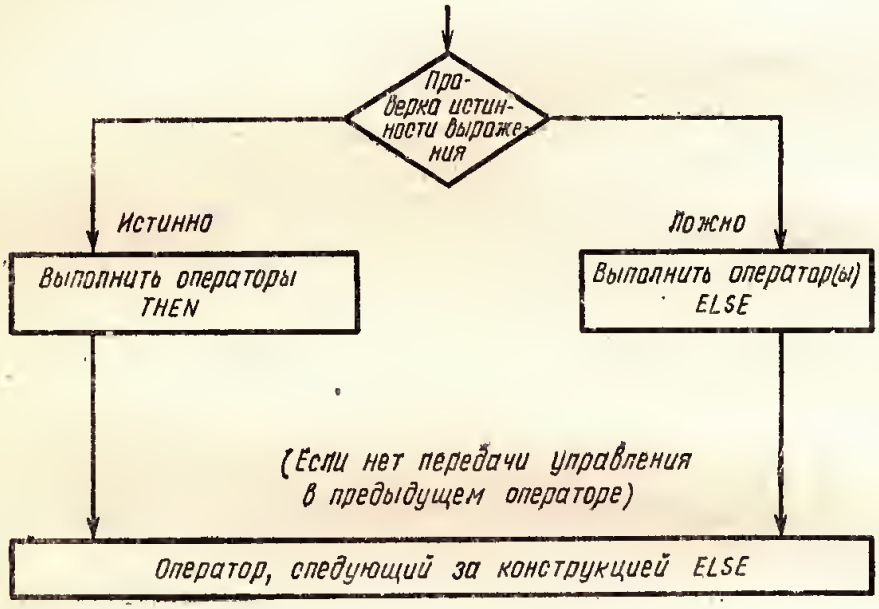


Рис. 2.2.

Пример

```
Фортран
  IF (A) 2, 2, 3
2 X=2.* A
  Y=A+X
  GO TO 4
3 X=3.* A
  Y=B+X
4 Z=X+Y
```

```
ПЛ/1
IF A <=0 THEN DO;
              X=2* A;
              Y=A+X; END;
         ELSE DO;
              X=3* A;
              Y=B+X; END;
Z=X+Y;
```

В конструкциях THEN и ELSE могут быть другие операторы IF. Например, оператор

```
IF A > B THEN IF C < D THEN GO TO INCREMENT;
```

построен правильно. Его можно записать следующим образом:

```
IF A > B & C < D THEN GO TO INCREMENT;
```

Арифметический оператор IF Фортрана мог бы быть записан на языке ПЛ/1 следующим образом:

```
IF A < B THEN GO TO LESS___THAN;
        ELSE IF A > B THEN GO TO GREATER___THAN;
                ELSE GO TO EQUAL;
```

Когда операторы IF вкладываются друг в друга таким образом, конструкция ELSE ассоциируется с вутренним оператором IF. Рассмотрим следующие вложенные операторы IF:

```
IF A=B THEN
  IF C < D THEN
    IF E > F THEN X=Y;
            ELSE Z=W;
      ELSE X=W;
C=D+E;
```

В этом примере первая конструкция ELSE связана с конструкцией IF E > F, а вторая конструкция ELSE связана с IF C < D. Это ил-

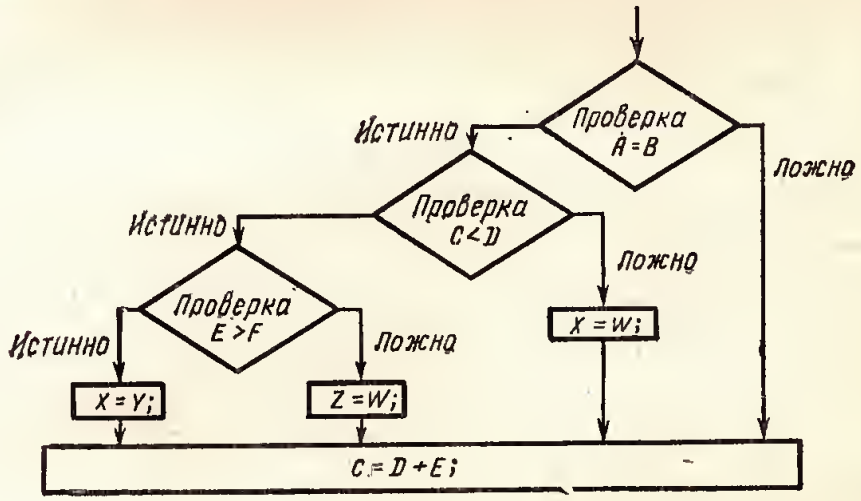


Рис. 2.3.

люстрирует блок-схема, представленная на рис. 2.3. Если логика программы требует, чтобы конструкции ELSE относились к первому и второму IF, то нужно вставить пустой оператор ELSE следующим образом:

```
IF A=B THEN
  IF C < D THEN
    IF E > F THEN X=Y;
            ELSE;
        ELSE Z=W;
    ELSE X=W;
C=D+E;
```

Это иллюстрирует блок-схема на рис. 2.4.

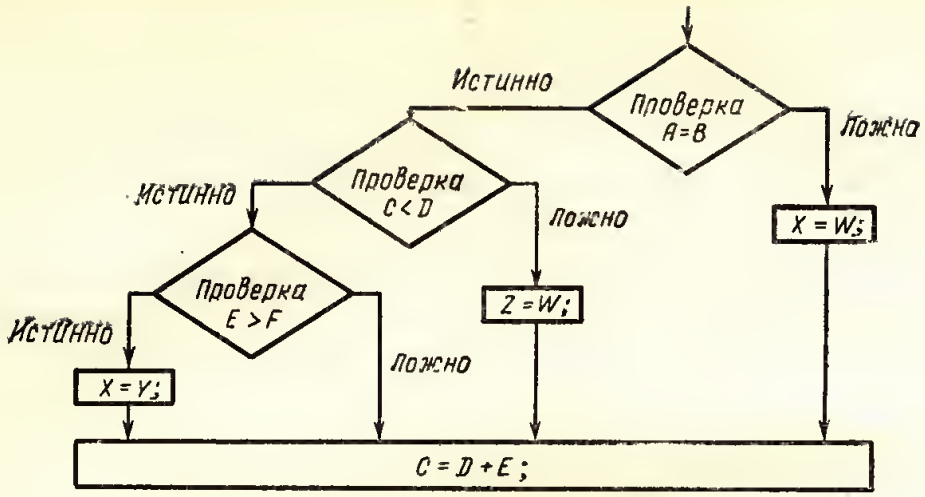


Рис. 2.4.

2.7. СПИСОК ВВОДА-ВЫВОДА

В ПЛ/1 существует несколько видов операций ввода-вывода. В этом разделе мы рассмотрим только LIST I/0 с системой ввода с перфокарт и системой вывода на построчное печатающее устройство.

INPUT (Ввод данных). Общая форма оператора ввода имеет вид:

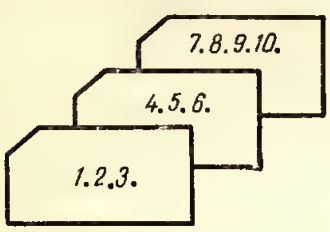
необязательная метка (метки): GET LIST (список имен переменных, разделенных запятымн);

Пример оператора GET LIST

```
GET LIST (A);
GET LIST (A, B  C);
```

Все 80 колонок перфокарты считываются одна за другой в непрерывном потоке. Данные перфорируются как последовательность констант, разделенных друг от друга по крайней мере одним пробелом или одной запятой. Внутри констант пробелы не допускаются. Сканирование вводимых данных продолжается до тех пор, пока список не будет исчерпан.

Например, если GET LIST (A, B); необходимо выполнить в программе пять раз, то все десять значений A и B можно отперфорировать на одной и той же карте, на десяти разных картах, по два числа на пяти перфокартах или любым другим методом. Очередность, в которой данные занесены на перфокарты, очень важна, так как эта очередность определяет, какой переменной приписываются эти значения. Если карты организованы так, как это показано на рис. 2.5, то ввод данных производится следующим образом:



Время выполнения	Значение A	Значение B
1	1.	2.
2	3.	4.
3	5.	6.
4	7.	8.
5	9.	10.

Рис. 2.5,

Оператор GET LIST (A) считывает десять чисел, если размерность A в операторе DECLARE определена A (10). Если размерность A определена A (5, 10), то считывается 50 элементов. Индексированные переменные заносятся в той последовательности, в какой они хранятся в памяти машины, т. е. быстрее всего изменяется последний индекс. Заметьте, что этот порядок прямо противоположен порядку, принятому в большинстве трансляторов Фортрана. Как и в Фортране, в ПЛ/1 список входных данных может задаваться с помощью циклов. Например,

```
GET LIST ((A(R) DO R=1 TO 50));
GET LIST (((A(K  J) DO K=1 TO 5) DO J=1 TO 10));
```

Заметьте, что в последнем примере порядок считывания аналогичен порядку считывания данных в Фортране.

```
GET LIST (M, N((A(L, J) DO J=I TO M) DO L=N TO I BY-1));
GET LIST (((A(I  J) DO I=I TO J) DO J=1 TO 10));
```

Список данных в операторах GET или PUT должен быть заключен в круглые скобки. У начинающих программистов наиболее распространенная ошибка заключается в том, что они опускают одни круглые скобки в примерах типа

```
GET LIST ((A(I) DO I=1 TO N));
```

OUTPUT (Вывод данных). Общая форма оператора вывода имеет вид:

необязательная метка (метки): PUT LIST (список данных, который должен быть выдан на печать; каждый символ разделен запятой);

Список данных, который выдается на печать, может состоять из переменной, выражения или строки символов. Строка символов, выдаваемая на печать, должна открываться и заканчиваться апострофом. В список могут входить также имена массивов и циклы; они считываются из памяти машины в том же порядке, что и в операторе GET LIST.

PAGE (страница), SKIP (пропуск), LINE (строка). Команда LIST управляет расположением данных вывода, но в ограниченной степени. Выходные данные пересылаются непрерывным потоком и печатаются строка за строкой. Транслятор определяет нужный формат для этих данных. Число символов, которые могут быть напечатаны на одной строке, зависит от размера строки в данном устройстве. Существуют три вида форматов, употребляемых только для печати данных. Это PAGE (страница), SKIP (пропуск), LINE (строка). Они записываются между операторами PUT и LIST. Их функции сводятся к следующему.

PAGE показывает, что печать должна начаться с первой строки следующей страницы.

SKIP (w) показывает, что должно быть пропущено (w — 1) строк и что печать должна начаться на строке w. Индекс w может быть опущен и тогда используется просто SKIP. Это показывает, что печать должна начаться со следующей строки. Если $w \leqslant 0$, то печатающее устройство осуществит печать на предыдущей строке. Если значение

w превышает число строк, оставшихся на данной странице, то будут пропущены все строки этой страницы и будет использована первая строка следующей.

LINE (w) показывает, что печать должна начаться со строки w текущей страницы. Если w = 0, то его полагают равным 1. Если печатающее устройство уже находится на строке w текущей страницы или за этой строкой, то пропускается вся страница, и печать производится на первой строке следующей. Возврат на предыдущие строки в устройствах построчной печати невозможен.

Заметьте, что формат SKIP определяет количество пропущенных строк, начиная с данного положения печатающего устройства, а формат LINE — номер строки текущей страницы.

Значение w может быть как константой, так и переменной, или выражением, которые могут быть преобразованы в целые десятичные числа до того, как будут выполняться SKIP или LINE.

(В подмножестве языка ПЛ/1 значение w должно быть константой без знака. В случае употребления LINE w должно быть меньше 256, а если оно равно нулю, то его полагают равным единице. Если w используется с форматом SKIP, то оно должно быть 0, 1, 2 или 3.)

Если в списке выходных данных есть двоичные числа, то они сначала преобразуются в десятичные числа с фиксированной или плавающей точкой (в зависимости от их внутреннего представления), а затем печатаются в виде десятичных чисел.

Следующие примеры иллюстрируют применение оператора PUT LIST:

```
        PUT LIST (A);
ABC:    PUT LIST (A, B, C);
        PUT PAGE LIST ('THE ANSWER IS', X);
        PUT SKIP (2) LIST (((A(I, J) DO I=1 TO 10) DO J=3 TO 5));
PRINT: PUT PAGE LINE (6) LIST ('HEADING');
```

Для иллюстрации применения форматов PUT, SKIP и LINE рассмотрим следующую процедуру:

```
T1: PROCEDURE OPTIONS (MAIN);
    DECLARE N FIXED BINARY REAL (15) INITIAL (5),
            X FLOAT DECIMAL REAL (6) INITIAL (2);
    PUT PAGE;
    PUT PAGE LIST ('A');
    PUT SKIP LIST ('B');
    PUT SKIP (3) LIST ('C');
    PUT SKIP (N) LIST ('D');
    PUT SKIP (N+X) LIST ('E');
    PUT PAGE LINE (30) LIST ('F');
    PUT SKIP (0) LIST ('G');
    PUT LINE (30) LIST ('H');
    PUT LIST ('I');
    END T1;
```

При выполнении этой процедуры происходит следующее.

1. Большинство трансляторов организуют печать выходных данных с первой строки следующей страницы. Допустим, что у нас именно этот случай. Команда PUT PAGE; установит бумагу на первой строке следующей страницы.

2. Перейти к первой строке новой страницы и напечатать A.
3. Перейти к следующей строке и напечатать B.
4. Пропустить три строки и напечатать C, т. е. оставить пропущенными две строки, прежде чем напечатать C.
5. Пропустить пять строк и печатать D, т. е. оставить пропущенными четыре строки.
6. Пропустить семь строк и печатать E, т. е. оставить пропущенными шесть строк.
7. Перейти к 30-й строке на следующей странице и печатать F.
8. Не допускать прогона бумаги перед печатью, т. е. напечатать на последней строке G.
9. Перейти к первой строке следующей страницы и напечатать H. При выполнении предыдущего шага мы находились на строке 30.
10. Напечатать I в следующем за H поле на той же строке, что и в шаге 9.

COPY (Скопировать). Конструкция COPY может быть добавлена к команде ввода данных GET LIST. Она обеспечивает выдачу входных данных без преобразований на стандартное печатающее устройство. На каждой строке печатается одно значение. Например, команда GET LIST (A, B, C) COPY; обеспечит запись значений A, B и C в том виде, в котором оно было воспринято с носителя на устройстве ввода. В примерах, которые будут даны далее, конструкция COPY будет рассмотрена с целью показать, как будут использоваться при выдаче на печать данные ввода. (Конструкция COPY не допускается в подмножестве языка).

Все операторы и конструкции, которые мы до сих пор рассматривали, теперь можно показать на примере законченных программ, записанных на языке ПЛ/1. Но прежде чем это сделать, следует остановиться еще на двух конструкциях, которые окажутся необходимыми при написании программ. Это два оператора ПЛ/1 нового типа. В полном объеме они будут рассмотрены в одной из следующих глав, и их употребление будет проиллюстрировано на примерах.

Оператор ON ENDFILE (при конце файла). Общая форма оператора имеет вид:

```
ON ENDFILE (имя файла) оператор ПЛ/1;
```

Оператор ПЛ/1 выполняется при чтении последней записи данных в указанном файле. Положение оператора ON ENDFILE не учитывается, если он выполняется непосредственно перед считыванием последней записи данных. Программы, которые даны в настоящей книге, выполнялись на машине IBM-360 при использовании устройства чтения с перфокарт как стандартной системы ввода (SYSIN). (В подмножестве языка ПЛ/1 после имени файла в качестве оператора берется GO TO.)

CHECK (Контроль). Существует прекрасный метод для проверки программы, записанной на языке ПЛ/1, когда во время ее выполнения возникают ошибки. При команде (CHECK (список идентификаторов, разделенных запятыми)) перед меткой оператора PROCEDURE можно получить листинг с указанием, как идет вы-

полнение программы. Если метка оператора — один из идентификаторов списка данных, то происходит прерывание как раз перед выполнением этого оператора и его метка выдается на печать. Если имя переменной — один из идентификаторов, то происходит прерывание всякий раз, когда меняется значение переменной и печатается идентификатор этой переменной и ее новое значение. В список данных для проверки могут быть внесены имена массивов.

Следующий пример простой, но законченной программы поможет понять многие из рассмотренных ранее принципов построения программ на ПЛ/1.

Программа считывает произвольный список чисел и считает их сумму. Результат печатается в стандартном формате, зависящем от устройства печати. В нашем случае при решении задачи печать данных всегда начинается с новой страницы. Для того чтобы обеспечить печать с начала новой страницы, необходима команда PUT PAGE LIST. Самые левые числа в листинге используются транслятором для диагностики.

```
            /* CHAPTER #2 EXAMPLE #2*/
            /* SAMPLE PROBLEM */
1  E202: PROCEDURE OPTIONS (MAIN);
2       DECLARE (SUM___OF___NUMBERS INITIAL(0), A) REAL FIXED DECIMAL (7,2);
3       ON ENDFILE (SYSIN) GO TO PUT;
5  INPUT: GET LIST (A);
6          SUM___OF___NUMBERS=SUM___OF___NUMBERS+A;
7          GO TO INPUT;
8  PUT: PUT LIST ('SUM IS ', SUM___OF___NUMBERS);
         /* NOTICE THAT PUT IS USED IN TWO WAYS */
9        END E202;
The data used were: 1 2 3 4 5 6 7 8 9 —2 —45 53 75
The sample output is
SUM IS                          126.00
```

Комментарий к программе

В начале программы

```
    /*Глава 2, пример 2*/
    /* Образец задачи*/
```

После оператора 8

```
    /* Заметьте, что PUT имеет два значения*/
```

Для того чтобы показать применение оператора CHECK в ПЛ/1, тот же самый пример выполняется с оператором

```
(CHECK (INPUT, SUM___OF___NUMBERS)):
```

который добавляется к процедуре:

```
            /* CHAPTER #2 EXAMPLE #3 */
            /* CHECK DEMONSTRATION */
1          (CHECK (INPUT, SUM___OF___NUMBERS)):
   E203:  PROCEDURE OPTIONS (MAIN);
2       DECLARE (SUM___OF___NUMBERS INITIAL (0), A) REAL FIXED DECIMAL (7,2);
3       ON ENDFILE (SYSIN) GO TO PUT;
5  INPUT; GET LIST (A);
6          SUM___OF___NUMBERS=SUM___OF___NUMBERS+A;
7          GO TO INPUT;
```

```
8  PUT: PUT LIST ('SUM IS', SUM _OF _NUMBERS);
        /* NOTICE THAT PUT IS USED IN TWO WAYS*/
9       END E203;
```

```
INPUT
SUM___OF___NUMBERS=          1 00;
INPUT
SUM___OF___NUMBERS=          3 00;
INPUT
SUM___OF___NUMBERS=          6.00;
INPUT
SUM___OF___NUMBERS=         10.00;
INPUT
SUM___OF___NUMBERS=         15 00;
INPUT
SUM___OF___NUMBERS=         21.00;
INPUT
SUM___OF___NUMBERS=         28.00;
INPUT
SUM___OF___NUMBERS=         36 00;
INPUT
SUM___OF___NUMBERS=         45.00;
INPUT
SUM___OF___NUMBERS=         43.00;
INPUT
SUM___OF___NUMBERS=        -2 00;
INPUT
SUM___OF___NUMBERS=         51.00;
INPUT
SUM___OF___NUMBERS=        126.00;
INPUT
SUM IS                          126.00
```

Комментарий к программе

В начале программы
```
    /*Глава 2, пример 3*/
    /* Демонстрация оператора CHECK*/
```
После оператора 8
```
    /* Заметьте, что оператор PUT имеет два значения*/
```

2.8. ПРИМЕРЫ ЗАКОНЧЕННЫХ ПРОГРАММ НА ПЛ/1

При ознакомлении со следующими примерами следует обратить внимание, как перфорируются операторы ПЛ/1. Каждая строка листинга предоставляет одну перфокарту. Перфорировать карты подобным образом необязательно, но это облегчает чтение программы. В частности, в этих целях в большинстве случаев оператор END, связанный с оператором DO, располагают непосредственно под соответствующим DO, а оператор ELSE, связанный с оператором THEN, — под соответствующим оператором THEN. Во всех случаях, кроме первого, значения входных данных печатаются с результатами вычислений как часть выходных данных. Все данные перфорируются в форматах, описанных в разделе «PUT LIST».

Пример А

Программа E204 выдает на печать таблицу значений квадратов и квадратных корней целых чисел от 1 до 10.

Заметьте, что собственно программа в основном состоит из одного оператора PUT. Эта программа выполняется на устройстве, позволяющем печатать пять выходных значений в строке. Количество чисел в строке определяется с помощью клавиш местного управления печатающего устройства. (Следует запомнить, что выходные данные представляют непрерывный поток информации.) Листинг программы и выходные данные приводятся далее.

Вид таблицы зависит от того, какой размер строки необходим для решения задачи и какое устройство определяет положение выходной таблицы. ПЛ/1 может осуществлять управление форматом выходных данных. Таблица, состоящая из трех столбцов, читается легче. Это иллюстрирует простая модификация примера А:

```
                /* CHAPTER #2 EXAMPLE #4 */
                /* SQUARE ROOT */
1  E204: PROCEDURE OPTIONS (MAIN);
2        DECLARE X FLOAT REAL DECIMAL (7);
3        PUT LIST ((X, X** 2, SQRT (X) DO X=1 TO 10));
4        END E204;
```

```
1.000000E+00   1.000000E+00   1.000000E+00   2.000000E+00   4.000000E+00
1.414213E+00   3.000000E+00   9.000000E+00   1.732050E+00   4.000000E+00
1.600000E+01   2.000000E+00   5.000000E+00   2.500000E+01   2.236067E+00
6.000000E+00   3.600000E+01   2.449489E+00   7.000000E+00   4.900000E+01
2.645751E+00   8.000000E+00   6.400000E+01   2.828427E+00   9.000000E+00
8.100000E+01   3.000000E+00   1.000000E+01   1.000000E+02   3.162277E+00
```

```
                /* CHAPTER #2 EXAMPLE #5 */
                /* SQUARE ROOT */
1  E205: PROCEDURE OPTIONS (MAIN);
2        DECLARE X FLOAT REAL DECIMAL (7);
3        DO X=1 TO 10;
4        PUT SKIP LIST (X, X** 2, SQRT (X));
5           END;
6           END E205;
```

```
1.000000E+00   1.000000E+00   1.000000E+00
2.000000E+00   4.000000E+00   1.414213E+00
3.000000E+00   9.000000E+00   1.732050E+00
4.000000E+00   1.600000E+01   2.000000E+00
5.000000E+00   2.500000E+01   2.236067E+00
6.000000E+00   3.600000E+01   2.449489E+00
7.000000E+00   4.900000E+01   2.645751E+00
8.000000E+00   6.400000E+01   2.828427E+00
9.000000E+00   8.100000E+01   3.000000E+00
1.000000E+01   1.000000E+02   3.162277E+00
```

Комментарии к программам

Заголовок первой программы

```
/* Глава 2, пример 4*/
/* квадратный корень*/
```

Заголовок второй программы

```
/* Глава 2, пример 5*/
/* квадратный корень */
```

Пример Б

В этом примере решается квадратное уравнение $Ax^2 + Bx + C = 0$.

Листинг исходной программы

```
          /* CHAPTER #2 EXAMPLE #6 */
          /* SOLUTION OF QUADRATIC EQUATION */
1  E206: PROCEDURE OPTIONS (MAIN);
2     DECLARE (A, B, C, DISC, E, X, ROOT1, ROOT2, F) REAL DECIMAL FLOAT (7);
3   ON ENDFILE (SYSIN) GO TO STOP;
5   PUT LIST ('SOLUTIONS OF QUADRATIC EQUATIONS');
6  IN: GET LIST (A, B, C);
7         PUT SKIP (2) LIST (A, B, C);
8         IF A=0 THEN IF B=0 THEN PUT SKIP LIST ('NOT AN EQUATION');
11                                ELSE DO;
12                                     X=-C/B; PUT SKIP LIST
                                              ('LINEAR EQUATION-ANSWER=',X);
14                                     END;
15                ELSE DO;
16                  DISC=B** 2-4*A*C; E=-B/(2*A);
18                  IF DISC=0 THEN DO; ROOT1, ROOT2=E;
21                                              PUT SKIP LIST (ROOT1, ROOT2); END;
23                                  ELSE IF DISC > 0 THEN DO; F=SQRT (DISC))/(2*A);
26                                            ROOT1=E+F; ROOT2=E-F;
28                                            PUT SKIP LIST
                                              (ROOT1, ROOT2);
29                                            END;
30                                  ELSE DO;
31                                            F=SQRT (-DISC)/(2*A);
32                                              PUT SKIP LIST (E, F);
33                                              PUT SKIP LIST (E, -F);
34                                              END;
35                  END;
36        GO TO IN;
37   STOP: END E206;
```

Комментарий к программе

Заголовок программы

```
    /* Глава 2, пример 6*/
/* Решение квадратного уравнения*/
```

Замечания

1. В программу специально не включены комплексные числа. Решение задачи с комплексными переменными дано далее.
2. Программа может решить произвольное число различных уравнений и затем прекращает вычисления. STOP в этой программе ПЛ/1 представляет собой метку оператора, а не команду.
3. Оператор 6 считывает значения коэффициентов, а оператор 7 печатает их.
4. Если A и B равны 0, то печатается сообщение и затем выполняется оператор 3 .
5. Если A = 0, а B ≠ 0, то, начиная с оператора 7, выполняется группа DO, а затем выполняется оператор 36.

6. Если A ≠ 0, то выполняется коиструкция ELSE группы DO. В этом случае примеияется формула квадратиого уравнения

$$X = \frac{-B}{2A} \pm \frac{\sqrt{B^2-4AC}}{2A}.$$

а) Группа DO, начиная с оператора 18, решает варнаит с двумя равными корнями. Отметьте, что в этой строке употребляется множествеиный оператор присваиваиия. (В подмиожестве ПЛ/1 употреблеиие множествеиных операторов присваивания не допускается.)

б) Начииая с оператора 23 группа DO решает варнант с двумя действительными неравиыми кориями.

в) С оператора 30 группа DO решает вариаит с сопряженными комплексиыми кориями.

7. Оператор 36 персдает управление оператору 6 для считывания коэффициентов нового уравнения. Если больше данных не поступает, то управление сразу же передается на оператор 37, и выполиеиие программы прекращается.

Вывод данных

Решение квадратных уравнений

```
0.000000E+00          0.000000E+00          2.000000E+00
NOT AN EQUATION

0.000000E+00          5.000000E+00          3.000000E+00
LINEAR EQUATION—ANSWER =                   -5.999999E-01

1.000000E+00          0.000000E+00         -1.000000E+00
1.000000E+00         -1.000000E+00

1.000000E+00         -2.000000E+00          1.000000E+00
1.000000E+00          1.000000E+00

1.000000E+00          0.000000E+00          1.000000E+00
0.000000E+00          1.000000E+00
0.000000E+00         -1.000000E+00

1.000000E+00         -2.000000E+00          2.000000E+0
1.000000E+00          1.000000E+00
1.000000E+00         -1.000000E+00
```

Замечанне

Вторая строка: не решенне
Четвертая строка: лииейиое уравиеиие — ответ =

Далее приводится модификация предыдущей программы с комплексными числами. Логическая схема построения программы аналогична только что рассмотренной. (В подмножестве ПЛ/1 комплексная арифметика не допускается.)

Листинг исходной программы

```
                /* CHAPTER #2 EXAMPLE #7 */
               /* SOLUTION OF QUADRATIC EQUATION */
 1  E207: PROCEDURE OPTIONS (MAIN);
 2      DECLARE (A, B, C, DISC, E, X) REAL DECIMAL FLOAT (7),
                (ROOT1, ROOT2, F) COMPLEX DECIMAL FLOAT (7);
 3      ON ENDFILE (SYSIN) GO TO STOP;
 5      PUT LIST ('SOLUTIONS OF QUADRATIC EQUATIONS');
 6  IN: GET LIST (A, B, C);
```

```
 7      PUT SKIP(2) LIST (A, B, C);
 8      IF A=0 THEN IF B=0 THEN DO;
11                                  PUT SKIP LIST ('NOT AN EQUATION');
12                                  GO TO IN;
13                                  END;
14                             ELSE DO;
15                                  X=-C/B; PUT SKIP LIST
                                          ('LINEAR EQUATION-ANSWER=', X);
17                                  GO TO IN;
18                                  END;
19              ELSE DO;
20                  DISC=B** 2-4*A*C; E=-B/(2* A);
22                  IF DISC=0 THEN DO; ROOT1, ROOT2=E; GO TO OUTPUT; END;
27                             ELSE IF DISC > 0 THEN DO; F=SQRT (DISC)/(2*A).
30                                                   ROOT1=E+F; ROOT2=E-F;
32                                                   GO TO OUTPUT; END;
34                                              ELSE DO;
35                                                   F=SQRT (-DISC)/(2*A)*I;
36                                                   ROOT1=E+F; ROOT2=E-F;
38                                                   GO TO OUTPUT; END;
40                  END;
41  OUTPUT: PUT SKIP LIST (ROOT1, ROOT2);
42          GO TO IN;
43      STOP: END E207;
```

Комментарий к программе

Заголовок к программе

```
/* Глава 2, пример 7*/
/* Решение квадратного уравнения */
```

Замечания

1. При выполнении программы E207 данные те же, что и при выполнении варианта E206, описанного ранее.
2. OUTPUT в операторе 41 представляет собой метку, присвоенную программистом. Метка OUTPUT не имеет ничего общего с тем, что оператор 41 является также оператором, обеспечивающим вывод данных программы.
3. Начиная с оператора 34 группа DO становится частью программы, вычисляющей сопряженные комплексные корни. Заметьте, что выражение в правой части оператора 35 становится мнимым при умножении на 1. Цифра 1 перед символом I обязательна.

OUTPUT (Вывод данных)

Решение квадратных уравнений

```
0.000000E+00        0.000000E+00              2.000000E+00
NOT AN EQUATION
0.000000E+00        5.000000E+00              3.000000E+00
LINEAR EQUATION-ANSWER=                      -5.999999E-01
1.000000E+00        0.000000E+00             -1.000000E+00
1.000000E+00+0.000000E+001                   -1.000000+00+0.000000E+001
```

```
1.000000E+00        -2.000000E+00        1.000000E+00
1.000000E+00+0.000000E+001               1.000000E+00+0.000000E+001
1.000000E+00         0.000000E+00        1.000000E+00
0.000000E+00+1.000000E+001               0.000000E+00-1.000000E+001
1.000000E+00        -2.000000E+00        2.000000E+00
1.000000E+00+1.000000E+001               1.000000E+00-1.000000E+001
```

Замечания

1. Обратите внимание на форму печати комплексных констант.
2. Для мнимой части варианты с вещественными равными корнями и вещественными неравными корнями печатаются как комплексные числа с нулем.

Пример В

Программа E208 считывает парами положительные целые числа до тех пор, пока данные не перестанут поступать и для каждой пары:

а) печатает пару считанных чисел;

б) печатает все положительные целые числа, на которые делится первое число, т. е. находит делители первого числа;

в) определяет делители второго числа;

г) находит самое большое положительное целое число, которое является делителем для обоих чисел, т. е. находит наибольший общий делитель.

Листинг исходной программы

```
          /* CHAPTER #2 EXAMPLE #8 */
          /* GREATEST COMMON DIVISOR */
 1  E208: PROCEDURE OPTIONS (MAIN);
 2        DECLARE (M, MM, N, NN, II, GCD) FIXED DECIMAL REAL (5,0),
                  (I, IJK) BINARY FIXED REAL (15,0);
 3        ON ENDFILE (SYSIN) GO TO STOP;
 5        I=0B;
 6  LOOP: I=I+1B;
 7          GET LIST (M, N);
 8          PUT SKIP (4) LIST ('DATA CARD' ,I, ':' ,M, N);
 9          MM=M; NN=N;
11       A: DO IJK=1, 2;
12          PUT SKIP LIST ('DIVISORS OF' ,MM, ':');
13  DO II=1 TO MM/2. MM;
14       IF MOD (MM, II)=0 THEN DO;
16                              PUT LIST (II);
17                              IF MOD (NN, II)=0 THEN GCD=II;
19                             END;
20       END;
21       MM=N; NN=M;
```

```
23      END A;
24      PUT SKIP LIST ('GREATEST COMMON DIVISOR—  ',GCD);
25      GO TO LOOP;
26  STOP: END E208;
```

Комментарий к программе

Заголовок программы

```
/* Глава 2, пример 8*/
/* Наибольший общий делитель */
```

На рис. 2.6 приведена блок-схема для этой программы, а далее — вывод данных.

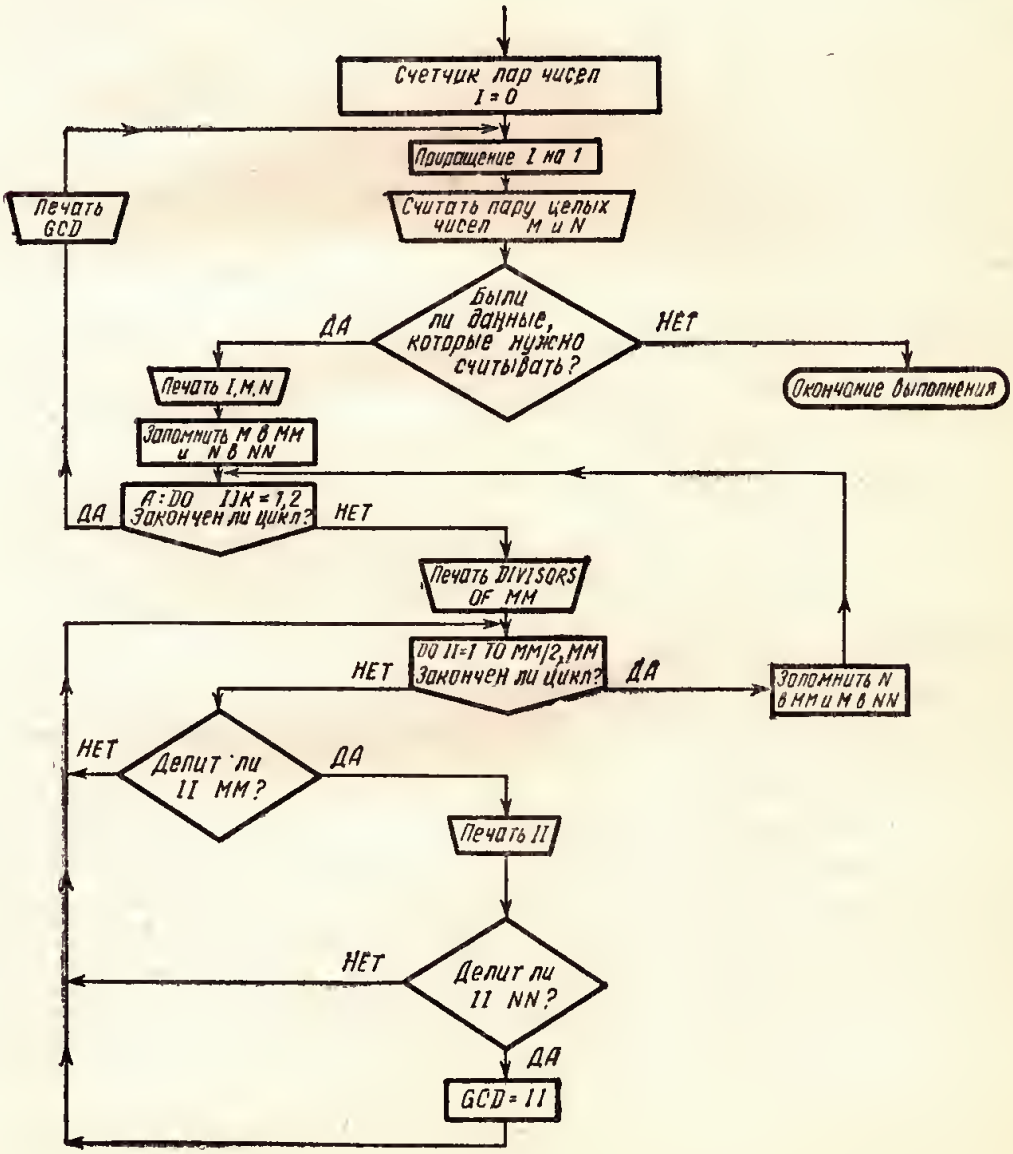


Рис. 2.6.

```
DA CARD                          1   :        12          8
DIVISORS OF                     12   :         1          2
  3                              4              6         12
DIVISORS OF                      8   :                    1          2
  4                              8
GREATEST COMMON DIVISOR—                                  4
DATA CARD                        2   :                    8         12
DIVISORS OF                      8   :                    1          2
  4                              8
DIVISORS OF                     12   :                    1          2
  3                              4              6         12
GREATEST COMMON DIVISOR—                                  4
DATA CARD                        3   :                   15         30
DIVISORS OF                     15   :                    1          3
  5                             15
DIVISORS OF                     30   :                    1          2
  3                              5              o         10         15
 30
GREATEST COMMON DIVISOR—                                 15
DATA CARD                        4   :                   30         15
DIVISORS OF                     30   :                    1          2
  3                              5              6         10         15
 30
DIVISORS OF                     15   :                    1          3
  5                             15
GREATEST COMMON DIVISOR—                                 15
DATA CARD                        5   :                   97       1053
DIVISORS OF                     97   :                    1         97
DIVISORS OF                   1053   :                    1          3
  9                             13             27         39         81
117                            351            1053
GREATEST COMMON DIVISOR—                                  1
```

Перевод текста

DATA CARD — данные, считанные с карты, DIVISOR OF — делители, GREATEST COMMON DIVISOR — наибольший общий делитель.

Пример Г

Программа на с. 64 намеренно дана с ошибками, для того чтобы показать, как важно быть крайне осторожным при объявлении переменных в программе.

В результате выполнения третьего оператора получается 0, а в результате работы четвертого — 33333. Эти операции выполняются так же, как и в Фортране*. Ситуация (NOFIXEDOVERFLOW) в операторе 7 до сих пор не рассматривалась. Она служит для предотвращения переполнения, что привело бы к прекращению выполнения программы. Выражения такого типа мы обсудим в главе 7. Переменные Y и Z не включены в оператор DECLARE. Следовательно, система определяет их по умолчанию. Это REAL DECIMAL FLOAT (6) (вещественное десятичное число с плавающей точкой (6)).

---

* Разные результаты при выполнении одной и той же операции над одними и теми же числами получаются потому, что форматы результатов различны. N — это целое двузначное число, а X — десятичное число с 14 знаками после десятичной точки. — *Примеч. пер.*

Правила по умолчанию будут рассмотрены в следующей главе.

Разница в результатах для Y и Z вызвана тем, что константы имеют точность, с которой они объявлены, в то время как формат результата деления типа FIXED определяется форматом аргумента, имеющим максимальную точность.

Значение Y получают следующим образом:

1 имеет точность (1, 0)

3 имеет точность (1, 0)

1/3 = 0.33333333333333 имеет точность (15,14)

25 имеет точность (2,0)

25 + 1/3 = 5.33333333333333 имеет точность (15,14) (отбрасывание производится с левой стороны)

Результаты Y хранятся в форме числа с плавающей точкой.

Значение Z получают следующим образом:

01 имеет точность (2,0)

3 имеет точность (1,0)

01/3 = 00.3333333333333

25 имеет точность (2,0)

25 + 01/3 = 25.3333333333333 имеет точность (15,13)

Результаты Z хранятся в форме числа с плавающей точкой.

Оператором 11 получено правильное значение, поскольку BIN было объявлено BINARY (двоичный), а десятичное число 1 было преобразовано в двоичное во время трансляции. Оператор 13 не требовал, чтобы константа была преобразована во время трансляции, он выдает тот же результат.

Комментарий к программе (на с. 64)

Заголовок программы

```
   /* Глава 2, пример 9*/
   /* Некоторые обычные ошибки*/
После оператора 2
   /* Усечение при делении целых чисел*/
После оператора 6
   /* Деление при получении результата с фиксированной запятой при неожиданном переполнении или усечении*/
После оператора 10
   /* Ошибка в присваивании идентификатора В двоичным константам */
```

Пример Д

Магический квадрат представляет собой квадратный массив с $n \times n$ отдельными положительными числами, основным свойством которых является то, что сумма n чисел, лежащих на любой горизонтальной, вертикальной или основной диагональной линии, всегда одинакова. В магическом квадрате $5^2 = 25$ элементов:

11	10	4	23	17
18	12	6	5	24
25	19	13	7	1
2	21	20	14	8
9	3	22	16	15

```
            /* CHAPTER #2 EXAMPLE #9 */
            /* SOME COMMON ERRORS */
 1 E209: PROCEDURE OPTIONS (MAIN);
 2            DECLARE N REAL DECIMAL FIXED (2), X REAL DECIMAL FIXED (15, 14),
               (BIN, ZIN) REAL BINARY FIXED (6) INITIAL (010011B);
   /* TRUNCATION IN INTEGER DIVISION */
 3            N=1/3;
 4            PUT LIST ('N=',N);
 5            X=1/3;
 6            PUT SKIP (3) LIST ('X=',X);
              /* FIXED POINT DIVISION RESULTING IN UNEXPECTED OVERFLOWS OR TRUNCATION */
 7 (NOFIXEDOVERFLOW): Y=25+1/3;
 8            PUT SKIP (5) LIST ('Y=', Y);
 9            Z=25+01/3;
10            PUT SKIP (3) LIST ('Z=', Z);
   /* FAILURE TO ADD IDENTIFIER "B" TO BINARY CONSTANTS */
11            BIN=BIN+1;
12
1             PUT SKIP (5) LIST ('BIN=', BIN);
13            ZIN=ZIN+1B;
14            PUT SKIP (3) LIST ('ZIN= ', ZIN);
15            END E209;
```

```
N=      0
X=      0.33333333333333
Y=      5.33333E+00
Z=      2.53333E+01
BIN=    20
ZIN=    20
```

```
          /* CHAPTER #2 EXAMPLE #10 */
          /* MAGIC SQUARE */
 1  E210: PROCEDURE OPTIONS (MAIN);
 2          DECLARE (A(3,3) INITIAL ((9)0), N INITIAL (3), INTEGER INITIAL (1), X, Y, PREVX, PREVY) REAL DECIMAL FIXED (2);
   /* PLACE THE INITIAL VALUE OF INTEGER IN THE MIDDLE ROW OF THE LAST COLUMN OF THE ARRAY */
 3          X=(N+1)/2;
 4          Y=N;
 5  NEXT:A(X, Y)=INTEGER;
   /* STORE THE ADDRESS OF THE LAST LOCATION FILLED IN PREVX AND PREVY */
 6           PREVX=X;
 7          PREVY=Y;
   /* INCREMENT THE VALUE OF INTEGER */
 8          INTEGER=INTEGER+1;
 9          IF INTEGER=10 THEN GO TO RESULTS;
   /* TO FIND THE NEXT LOCATION FOR INTEGER ADD ONE TO EACH OF THE PREVIOUS INDICES */
11          X=X+1;
12          Y=Y+1;
   /* IF X AND/OR Y ARE GREATER THAN N CHANGE THEIR VALUE (S) TO ONE*/
13          IF X>N THEN X=1;
15          IF Y>N THEN Y=1;
   /* CHECK TO SEE IF THE NEW LOCATION HAS BEEN FILLED IN A PREVIOUS STEP. IF NOT PLACE THE PRESENT VALUE OF
      INTEGER IN IT */
17          IF A(X, Y)=0 THEN GO TO PLACE;
   /* IF THE LOCATION IS OCCUPPIED PLACE THE VALUE OF INTEGER IN THE LOCATION REPRESENTATED BY A(PREVX, PREVY-1) */
19          ELSE DO;
20                  X=PREVX;
21                  Y=PREVY-1;
22          END;
23 PLACE:GO TO NEXT;
   /* PRINT THE MAGIC SQUARE */
24 RESULTS; PUT LIST ('THE MAGIC SQUARE SOLUTION FOR N=3');
25           PUT SKIP (3) LIST (A(1,*), ' ', ' ', A(2,*), ' ', ' ', A(3,*));
26          END E210;
```

THE MAGIC SQUARE SOLUTION FOR N=3

```
4   3   8
9   5   1
2   7   6
```

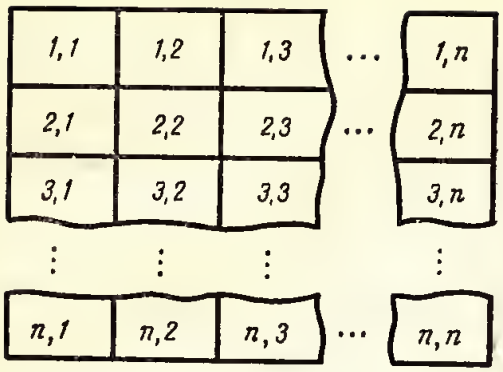


Рис. 2.7.

Сумма всех чисел в горизонтальных, вертикальных рядах и по диагонали равна 65. Для данного n существует много возможных решений. Далее описана процедура для конструирования магического квадрата только с *нечетными* значениями n. Для того чтобы иметь возможность показать место числа внутри квадрата, изобразим квадрат $n \times n$, как это показано на рис. 2.7.

Алгоритм для получения магического квадрата $n \times n$ для нечетных целых чисел лучше показать на блок-схеме. Приведенная на с. 65 программа написана для случая, когда $n = 3$.

Комментарий к программе (на с. 65)

Заголовок программы

```
    /* Глава 2, пример 10*/
    /* Магический квадрат*/
После оператора 2
    /* Поместить начальное значение целого числа в среднюю строку последнего
столбца массива*/
После оператора 5
    */ Запомнить адрес последней заполненной ячейки квадрата в PREVX и
PREVY*/
Перед оператором 8
    /* Приращение значения переменной INTEGER*/
После оператора 9
    /* Найти адрес следующей ячейки квадрата для переменной INTEGER,
прибавить 1 к каждому из предыдущих индексов*/
Перед оператором 13
    /* Если X и/или Y больше чем N, принять их равными 1*/
Перед оператором 17
    /* Проверить, заполнена ли новая ячейка квадрата при выполнении преды-
дущего шага. Если нет, то поместить в него текущее значение переменной
INTEGER*/
После оператора 17
    /* Если ячейка занята, поместить значение переменной INTEGER в ячейку
квадрата A (PREVX, PREVY — 1)* /
После оператора 23
    /* Выдать на печать магический квадрат* /
```

Блок-схема к этой программе представлена на рис. 2.8.

Пример E

Программа на с. 68 считывает матрицы A (2×3) и B (3×4) и печатает их произведение в виде матрицы C (2×4).

Матрица — это прямоугольный массив чисел. В нашей программе входными данными служат матрицы:

$$A = \begin{pmatrix} 2 & 4 & 6 \\ 3 & 5 & 7 \end{pmatrix} \quad B = \begin{pmatrix} 12 & 15 & 43 & 85 \\ 12 & 42 & 73 & 46 \\ 2 & 5 & 32 & 1 \end{pmatrix}$$

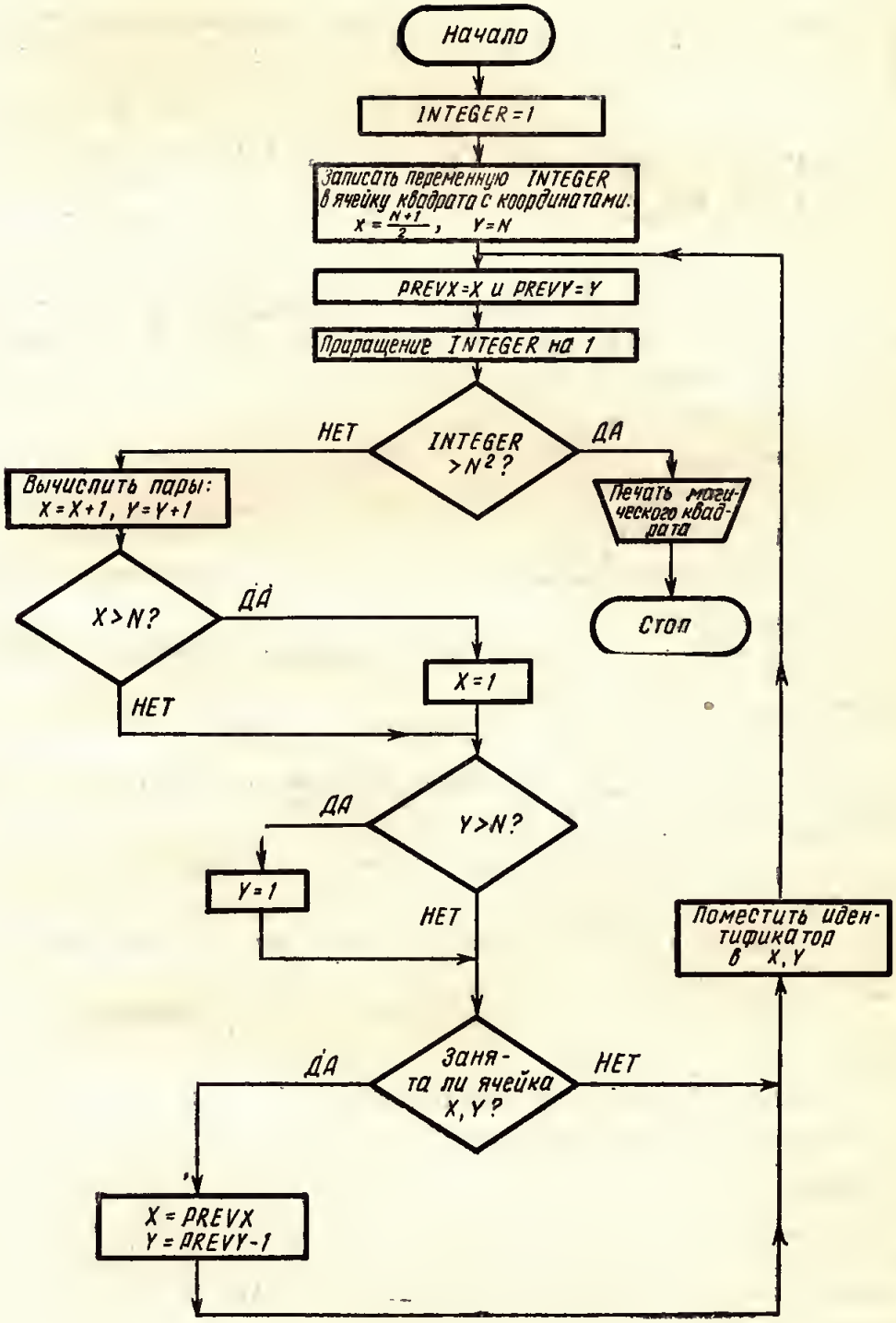

Начало
INTEGER=1
Записать переменную INTEGER в ячейку квадрата с координатами. X=(N+1)/2, Y=N
PREVX=X и PREVY=Y
Приращение INTEGER на 1
INTEGER >N²?
НЕТ
ДА
Вычислить пары: X=X+1, Y=Y+1
Печать магического квадрата
Стоп
X>N?
ДА
НЕТ
X=1
ДА
Y>N?
Y=1
НЕТ
Поместить идентификатор в X, Y
ДА
Занята ли ячейка X, Y?
НЕТ
X=PREVX
Y=PREVY-1

Рис. 2.8.

```
          /* CHAPTER #2 EXAMPLE #11 */

          /* MATRIX MULTIPLICATION*/

1  E211:PROCEDURE OPTIONS (MAIN);
2          DECLARE (A(2,3), B(3, 4)  C(2,4) INITIAL ((8)0)) REAL DECIMAL FIXED (5);
   /* READ VALUES OF MATRICES 'A' AND 'B'. */
3          GET LIST (A, B);
   /* MULTIPLY MATRIX 'A' BY MATRIX 'B' AND PLACE THE RESULTS IN MATRIX 'C'*/
4  LOOP:DO X=1 TO 2;
5           DO Y=1 TO 3;
6                DO Z=1 TO 4;
7                   C(X, Z)=A(X, Y)*B(Y, Z)+C(X, Z);
8          END LOOP;
   /* PRINT MATRIX 'C' */
11         PUT LIST ('THE PRODUCT OF THE 2X3 MATRIX ''A'' AND THE 3X4
             MATRIX ''B''');
12         PUT SKIP LIST ('IS THE 2X4 MATRIX ''C'' PRINTED BELOW');
13          PUT SKIP (3) LIST ((C(1,I) DO I=1 TO 4));
14         PUT SKIP LIST ((C(2,I) DO I=1 TO 4));
15         END E211;
```

```
THE PRODUCT OF THE 2X3 MATRIX ''A'' AND THE 3X4 MATRIX ''B''
IS THE 2X4 MATRIX 'C' PRINTED BELOW

   84        228        570        360
  110        290        718        492
```

Для того чтобы получнть элемент строки *i* столбца *j* в матрице произведения, необходимо вычнслить сумму пронзведенний соответствующих элементов строки *i* в матрнце A н столбца *j* в матрнце B. Например, число 718 находится во второй строке третьего столбца матрицы C. Для получения этого результата с элементами второй строки матрицы A (3 5 7) и третьего столбца матрнцы B (43 73 32) производятся следующие вычислення: (3) (43) + (5) (73) + (7) (32) = = 718.

Элементы матриц A и B считываются оператором 3. Необходнмо помнить, что последний нндекс изменяется быстрее другнх; данные перфорнруются на карту следующим образом: 2 4 6 3 5 7 12 15 43 85 12 42 73 46 25 32 1.

Коммеитарий к программе (на с. 68)

Заголовок программы

```
    /* Глава 2, пример 11*/
    /* Умиожеине матриц */
После оператора 2
    /* Считать зиачеиия матриц A и B*/
После оператора 3
    /* Умножнть матрицу А иа матрицу В и поместить результаты в матрицу С*/
После оператора 8
    /* Печать матрицы С*/
```

Перевод заголовка результата

Произведеиие матрицы А (2×3) и матрицы В (3×4) равио матрице С (2×4), приведеиной иижe.

2.9. УПРАЖНЕНИЯ

Упражиения, данные в этом разделе, помогут понять миогие новые идеи ПЛ/1, о которых говорилось в данной главе. Упражиения делятся на два раздела. Первый состоит из коротких вопросов, а второй — из четырнадцати задач для программирования. Прежде чем перейти к изучению следующей главы, необходимо выполиить первую группу упражнений полиостью, а затем написать несколько программ из второго раздела и, если это будет возможио, пропустить эти программы на вычислительиой машине.

По упражиеииям необходимо иаписать миииимум пять программ:
первая программа — упражнение 1 или 2;
вторая программа — упражнение 3, 4, 5, 6 или 7;
третья программа — упражиение 8, 9 или 10;
четвертая программа — упражнеиие 11, 12 или 14;
пятая программа — упражнение 13.

Короткие упражнения

1. Какие из указаиных меток пригодны для оператора PROCEDURE?

а) 27
б) PROBLEM___27
в) END
г) # 27
д) P27

2. Какне из указаиных идеитифнкаторов пригодиы для операторов ПЛ/1 и не пригодны в качестве меток для оператора PROCEDURE?

а) F (X)
б) START
в) DO AGAIN
г) THE___CALCULATED___VELOCITY___IS
д) SIN

3. Какие из следующих законченных операторов ПЛ/1 правильны? Если операторы содержат ошибки, попытайтесь их исправить.

а) IF X+Y<X−4 THEN GO TO BEGIN;
б) VELOCITY=RATE* TIM;
в) START;END;
г) ADD: A+B=C;
д) DO C=1, 10 TO D* E−4;
е) DISC=B** 2−4A* C;
ж) PUT SKIP (4) LIST X, Y, Z;
з) CHECK (SUM, X, Y); #1; PROCEDURE OPTIONS MAIN;
и) ELSE;
к) $ 10.00: CHANGE___FOR___TEN=10.00−COST;
л) A, B, C=A+B+C;

4. Запишите в наиболее простой форме оператор ПЛ/1, который установит X = 5, если A и B имеют один и тот же знак, или X = 4, если A и B имеют противоположные знаки или равны 0.

5. Проанализируйте следующие сегменты программы:

```
DECLARE (A FIXED (2), B FIXED (4,2), C FLOAT (7)) REAL DECIMAL;
GET LIST (A, B, C);
```

а) Если на карте отперфорированы данные:

```
12345  12345  12345
```

то какое значение примут переменные A, B, C после выполнения оператора GET?

б) Какое значение примут переменные A, B, C после выполнения оператора GET, если данные отперфорированы следующим образом:

```
·12345  ·12345  ·12345
```

6. Проанализируйте следующий сегмент программы:

```
DECLARE (A FIXED (4), B FIXED (6,2)) BINARY REAL;

GET LIST (A, B);
```

а) Какие значения примут переменные A и B после выполнения оператора GET, если данные на карте отперфорированы так:

```
15  7.75
```

б) Если сразу же за оператором GET следует оператор PUT LIST (A, B); что будет выдано на печать?

в) Ответьте на вопрос пункта а), если данные отперфорированы на карте 20
20

г) Ответьте на вопрос пункта б), если данные отперфорированы на карте 20

7. Сколько строк выходных данных будет отпечатано следующими сегментами программы:

а)
```
DO X=1 BY .5 TO 5;
  Z=SQRT (X);
  PUT LIST (Z);
  END;
```

б)
```
DO X=1 BY .5 TO 5;
  DO Y=10 TO 7 BY−1;
  DO Z=2 TO 1 BY−.1;
  A=SQRT (X+Y+Z);
  PUT SKIP LIST (A);
  END; END; END;
```

в)
```
DO X=1 TO 100 WHILE (Z<=5);
  Z=SQRT (X);
  PUT SKIP LIST (Z);
  END;
```

8. Расставьте круглые скобки в следующих выражениях ПЛ/1, чтобы показать последовательность их выполнения:

а) A**B* 2+3.456*Y
б) A+B*C−D/F
в) A=B | A<C &—| (B—|<C** 2)
г) A**B**C−D=A* B/C

9. Какие круглые скобки в приведенных выражениях можно снять, не изменив значения этих выражений?

а) (A+B)/C
б) A+(B/C)
в) —| (—| (A<B) | (C>=D))
г) (A+B)−(C+D)+2* E
д) (A+B)+((C+D)* 2* E)

10. Нарисуйте блок-схемы, объясняющие употребление следующих IF:

а) IF A=B THEN IF C=D THEN W=X; ELSE W=Y; ELSE W=Z;
б) IF A=B THEN IF C=D THEN W=X; ELSE W=Y;
в) IF A=B THEN IF C=D THEN W=X; ELSE; ELSE W=Y;
г) IF A=B THEN W=X; ELSE IF C=D THEN W=Y; ELSE W=Z;

11. Если A = 1, B = 6, а D=8, какие из приведенных выражений будут истинными?

а) (A<B) | (D=C)
б) —| (A>B) & (D>C)
в) A>B &—| D>C
г) A<B | A>=B & A=A
д) 2* A+B=D | D−C<B−A

12. Написать сегмент программы чтения произвольного списка чисел; сумму положительных чисел хранить в SUM_POS, сумму отрицательных—в SUM_NEG; подсчитать, сколько нулей содержится в списке.

13. Если вам нужно записать программу, которая будет считывать оценки двадцати пяти студентов во время приема экзаменов, вычислять их средний балл, а затем выдавать на печать их оценки и средний балл, каким образом вы объявите идентификаторы оценок и среднего балла? Допустим, что оценки будут определяться целыми числами от 0 до 100.

14. Допустим, что программа сформировала список из N чисел в возрастающем порядке. Они хранятся в A(1), A (2), ..., A (N). Напишите сегмент программы таким образом, чтобы выдать эти числа на печать, каждое на отдельной строке, начиная с новой страницы, но в убывающем порядке.

Задачи для программирования

1. Напишите программу для вычисления объема шара. Величина радиуса считывается с перфокарты данных, а печатаются радиус и вычислительный объем. (Объем шара равен $4.1888R^3$.)

2. Напишите программу нахождения гипотезы любого прямоугольного треугольника. Значения длин двух сторон считываются с перфокарты данных. Эти значения и вычисленное значение гипотенузы выдаются на печать. (Квадрат гипотенузы прямоугольного треугольника равен сумме квадратов катетов.)

3. Пары чисел, x и y, отперфорированы на картах данных. О том, сколько пар чисел должно быть считано, никакой информации нет. Предусмотрите чтение этих чисел и вычисление суммы всех значений x, суммы всех значений y, суммы квадратов значений x, суммы произведений каждой пары x и y, среднее арифметическое значений x и среднее арифметическое значений y. Среднее арифметическое находят делением суммы чисел на количество чисел. Хранить все значения x и y не нужно. Выдайте на печать все указанные результаты.

4. Напишите программу чтения списка чисел и предусмотрите выдачу на печать самого большого и самого маленького числа в этом списке. О том, сколько чисел содержится в списке, нет никакой информации, за исключением того, что есть по крайней мере одно число.

5. Напишите программу вычисления π до 15-го десятичного знака.

Значение π можно получить с любой требуемой степенью точности с помощью следующего ряда:

$$\pi = 6\left[\frac{1}{2} + \left(\frac{1}{2}\right)\left(\frac{1}{3}\right)\left(\frac{1}{2}\right)^3 + \left(\frac{1}{2}\right)\left(\frac{3}{4}\right)\left(\frac{1}{5}\right)\left(\frac{1}{2}\right)^5 + \right.$$
$$\left. + \left(\frac{1}{2}\right)\left(\frac{3}{4}\right)\left(\frac{5}{6}\right)\left(\frac{1}{7}\right)\left(\frac{1}{2}\right)^7 + \ldots\right] = 3 + \left(\frac{3}{2}\right)\left(\frac{1}{3}\right)\left(\frac{1}{2}\right)^3 +$$
$$+ \left(\frac{3}{2}\right)\left(\frac{3}{4}\right)\left(\frac{1}{5}\right)\left(\frac{1}{2}\right)^5 + \left(\frac{3}{2}\right)\left(\frac{3}{4}\right)\left(\frac{5}{6}\right)\left(\frac{1}{7}\right)\left(\frac{1}{2}\right)^7 + \ldots$$

Обратите внимание на то, что в этой формуле наблюдается следующая закономерность:

второй член равен первому члену, умноженному на $(1/2)(1/3)\ (1/2)^2$;

третий член равен второму члену, умноженному на $(3/4)(3/5)(1/2)^2$;

четвертый член равен третьему члену, умноженному на $(5/6)(5/7)(1/2)^2$;

пятый член равен четвертому члену, умноженному на $(7/8)(7/9)(1/2)^2$ и т. д.

(Будьте очень внимательны при вычислении элементов ряда.)

6. Последовательность чисел Фибоначчи записывается следующим образом: 1, 1, 2, 3, 5, 8, 13, ..., где каждое следующее число после первых двух является суммой двух предыдущих чисел. Напишите программу для вычисления и печати списка по крайней мере двадцати чисел этой последовательности.

7. (Составление таблиц счетов вкладчиков). Напишите программу, которая будет считывать данные в следующем порядке: сумма вклада в долларах и центах; годовой процент, выраженный десятичным числом; целое число, показывающее, сколько раз в году выплачивался указанный процент; год внесения вклада; первый и последний годы, за которые необходимо выдать баланс на печать; затем выдать на печать таблицу со всей требуемой информацией. Например, если данные таковы: 10.00.05 4 1900 1960 1980, то необходимо выдать на печать таблицу данных с 1960 по 1980 г., показав ежегодный баланс суммы в 10.00 долларов, вложенной в 1900 г. при 5%, выплачиваемых ежеквартально.

8. Эратосфен, греческий философ (230 г. до н.э.), составил алгоритм для нахождения простых чисел. Этот метод называется «решетом Эратосфена». Простым числом называется целое число, которое не делится ни на одно число, кроме единицы или самого себя. Чтобы найти все простые числа менее или равные 100 методом «решета», необходимо:

а) составить следующий список чисел: 1 2 3 5 7... 99 (все нечетные целые числа $<$ 100 после 2);

б) исключить каждое третье число списка после цифры 3;

в) исключить каждое пятое число после цифры 5, если оно не было исключено ранее;

г) исключить каждое седьмое число после цифры 7, если оно не было исключено раньше;

д) продолжить этот процесс.

Оставшиеся числа простые. Напишите программу с помощью этого алгоритма для нахождения и выдачи на печать всех простых чисел, меньших или равных 1000.

9. Напишите программу, которая будет считывать произвольное количество пар чисел. Первое число каждой пары показывает сумму счета (в долларах и центах), второе — сколько по этому счету уже уплачено. Общая сумма счета не должна превышать 10 000 долларов. Для каждой пары чисел необходимо выдать на печать:

а) сообщение ВСЯ СУММА ОПЛАЧЕНА (THE CORRECT AMOUNT WAS PAID), если счет оплачен верно, или

б) сумму, которую следует оплатить (в долларах и центах), если оплачен не весь счет, или

в) сколько необходимо выдать сдачи, если уплаченная сумма превышает сумму счета. В каких купюрах и монетах лучше сдать сдачу (при наименьшем количестве купюр и монет), если имеются бумажные деньги достоинством в двадцать долларов, десять долларов, пять долларов и один доллар, а монеты — достоинством в двадцать пять центов, десять центов, пять центов и один цент. Например, если необходимо сдать сумму в 1 доллар и 26 центов, то должно быть напечатано, что необходимо вернуть одну однодолларовую бумажку, одну двадцатипятицентовую монету и одну одноцентовую монету. Нельзя выдавать 5 двадцатипятицентовых монет и 1 монету в один цент или деньги в каких-нибудь других комбинациях.

10. Напишите программу для конвертирования долларов США в марки ФРГ, и наоборот. Ввод данных в программу производится в форме пар чисел. Если первое число пары — 1, то второе число показывает количество долларов, которое необходимо конвертировать в марки. Если первое число равно +1, то второе число показывает количество марок, которое следует конвертировать в доллары. Воспользуйтесь следующим соотношением: 1 марка = 0,2524 доллара.

11. Составьте таблицу синуса. Входные данные должны состоять из двух десятичных чисел, которые показывают требуемый интервал; первое число показывает угол в градусах и долях градуса первого значения таблицы, а второе — угол в градусах и долях градуса ее последнего значения. Разделите интервал от начального до конечного числа на 100 равных промежутков и обеспечьте печать таблицы с указанием угла в градусах, минутах и секундах, а также синус этих углов. (*Замечание.* Встроенная функция SIN дает синус угла в радианах и долях радиана, в то время как встроенная функция SIND подсчитывает синус угла в градусах и долях градуса.)

12. Случайные числа находят большое применение в статистике и моделировании. Идеальный генератор случайных чисел не должен повторять цикла независимо от того, как долго он работает. Псевдогенератор случайных чисел может давать относительно большой цикл повторения, длина которого зависит от требований решаемой задачи. Задача программиста — уравновесить усилия, необходимые для получения случайного числа с длиной цикла повторения. Существуют много способов для получения псевдослучайных чисел. Далее дается описание алгоритма для получения ряда четырехзначных псевдослучайных целых чисел. Вначале берем два произвольных четырехзначных целых числа. Одно из них (назовем его RAND) и будет первым случайным числом ряда. Второе (назовем его CONSTANT) будет взято для получения других случайных чисел. Для того чтобы вычислить второе случайное число, необходимо умножить число RAND на число CONSTANT, разделить полученное произведение на 10 и опустить цифру, стоящую после запятой. Четыре цифры в младших разрядах полученного числа и будут новым случайным числом.

Пример

```
          RAND=1306
      CONSTANT=7829
(RAND)*(CONSTANT)=10224674
```

Разделим это число на 10 и опустим цифру после запятой; в итоге получим 1022467. Четыре последние цифры дают число 2467.

Для получения следующего случайного числа необходимо повторить описанный процесс, взяв то же самое число CONSTANT, а в качестве числа RAND — только что вычисленное случайное число.

Напишите программу, которая будет считывать два целых числа N и CONSTANT, где N означает ряд случайных чисел, которые должны быть вычислены, $N \leqslant 500$; а CONSTANT — число, которое необходимо для получения всех случайных чисел. За первое случайное число примите цифру 1306. После вычисления ряда случайных чисел N, выдайте их на печать. Затем определите количество вхождений каждой цифры от 0 до 9 в полученные случайные числа. Обеспечьте печать таблицы, показывающей это распределение.

Это не самый лучший алгоритм для получения случайных чисел. Далее будут рассмотрены другие методы.

13. Напишите программу, которая будет считывать список из двух или более чисел, а затем сортировать их и печатать в возрастающем порядке. Допустим (с целью использования оператора DECLARE), что список будет считываться до

тех пор, пока данные будут содержать ие более 100 чисел. Однм из алгоритмов сортировки может быть следующий.

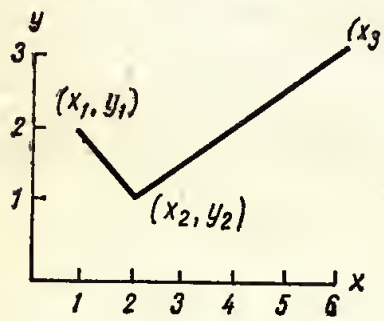
Сравнм два первых числа и, если это необходимо, поменяем их местами так, чтобы второе число было больше первого. После проведения этой операцяи сравним второе и третье числа списка, и, если это необходимо, опять помеияем их местами, чтобы наибольшее число стояло иа третьем месте. Повторим эту операцию с третьим и четвертым чнсламн и т. д. После того как мы пройдем подобиым образом весь список, в коице списка окажется самое большое число. Затем повторим всю операцию с самого начала, ие включая в повторение последнее число списка, так как уже установлеио, что оно является самым большим в данном списке. Во время этой операции мы устаиовим предпоследиее чнсло спнска. Процесс сравиеняя будет проводяться до тех пор, пока остаиутся только два числа. И как только онн будут отсортироваиы, составлеиие списка в возрастающем порядке будет закоичено.

В приведениый алгоритм легко внести некоторое усовершенствоваиие. Если необходимо отсортяровать список в сотию чисел, то полученне списка этих чисел в возрастающем порядке с помощью метода, опясаииого выше, потребует девяиосто девять просмотров. Но может произойти так, что сортировка закоичится, иапример, на пятом просмотре и что инкаких перестаиовок чисел больше не потребуется. Если полученная последовательиость удовлетворяет программу, то процедура сортнровки может быть закончена иа этой стадии. Добавьте такую проверку к программе при каждом прохождеиии через список чисел.

В следующих главах будут рассмотрены другие алгоритмы сортировки.

14. Электронная вычнслительиая машииа прямеияется для числового нитегрирования. Однн из простых типов такого интегрироваиия — вычясление площади, ограниченной сверху кривой. Напншнте программу, которая будет считывать не меиее 2 и ие более 500 точек (x, y) такой кривой. Входиые даиные упорядочены по возрастаиию значения x. Вычислять площадь следует по формуле трапеций. Площадь трапецнн равна полусумме двух параллельных сторон, умиожеиной иа высоту. Трапеции образуются в результате соедниения точек прямымн линиями.

Пример.



(x_3, y_3) (x_1, y_1), (x_2, y_2), (x_3, y_3). Данные 1 2 2 1 6 3.

Рнс. 2.9.

$$\text{Площадь} = \frac{1}{2}(y_1 + y_2)(x_2 - x_1) + \frac{1}{2}(y_2 + y_3)(x_3 - x_2) =$$

$$= \frac{1}{2}(2+1)(1) + \frac{1}{2}(1+3)(4) = \frac{3}{2} + 8 = \frac{19}{2}.$$

Глава 3

ОСНОВНЫЕ КОНСТРУКЦИИ ЯЗЫКА

3.1. СИНТАКСИС

В последующих главах много места уделяется детальному описанию языка ПЛ/1 и возможностям его применения, в том числе методам обработки данных. Формальное описание языка — предмет совсем не новый. Западные языки изучаются со времен Древнего Рима. Как правило, эти языки содержат набор символов, называемый алфавитом. Буквы алфавита группируются в слова, а слова организуются в предложения. В каждом языке существуют различные правила образования предложений и более сложных соединений слов. Синтаксис языка — это набор правил, по которым слова организуются в предложения. Грамматика языка имеет дело с классами слов и их отношениями друг с другом. Однако можно организовать слова точно в соответствии с правилами синтаксиса и грамматики, а предложение не будет иметь смысла.

Точное понимание значений слов и языковых структур обеспечивается семантикой языка.

Язык ПЛ/1 создан для связи человека с ЭВМ. Операторы ПЛ/1 должны быть переработаны в код команд машины. Для генерирования программы в кодах команд транслятор ПЛ/1 использует правила синтаксиса этого языка. В некоторых случаях в процессе трансляции с помощью семантики обеспечивается правильность рабочей программы, даже если в исходной программе имелись неточности. Однако для любого транслятора исправление ошибок исходной пограммы — трудная задача. Для точного описания синтаксической структуры ПЛ/1 удобна некоторая система обозначений. В процессе создания машинных языков было развито несколько таких систем. Одна из наиболее известных — нормальная форма Бэкуса. Эту систему символов иногда называют *метаязыком**. Такая система обозначений не явля-

* Метаязыком называется язык, предназначенный для описания грамматики какого-либо языка. — *Примеч. пер.*

ется частью языка программирования. Она пригодна только для точного описания конструкций, применяемых в описываемом языке. Система обозначений, принятых в настоящей книге, приведена далее. Необходимо помнить, что эта система может показывать только порядок организации элементов, пунктуацию, которую можно применять, и т. п. Но с ее помощью нельзя описать смысл. Для его понимания необходимо знать семантику текста, написанного на языке ПЛ/1.

Синтаксические обозначения. 1. В фигурные скобки { } заключается несколько выражений, из которых нужно сделать выбор. Внутри скобок эти выражения могут быть расположены вертикально или горизонтально. В последнем случае каждое выражение должно быть отделено друг от друга вертикальной чертой:

$\left\{\begin{matrix}\text{stream}\\\text{record}\end{matrix}\right\}$ или {stream | record}

2. В квадратные скобки [ ] заключаются произвольные операторы или выражения:

```
[label:]
NOLABEL] [VERIFY]
```

3. Три точки ... указывают на повторение предыдущей конструкции один или несколько раз.

3.2. ЭЛЕМЕНТЫ ОРГАНИЗАЦИИ ПРОГРАММЫ

Набор символов. Основной элемент языка — набор символов. В языке ПЛ/1 два алфавита: из 48 символов и из 60 символов. Если устройства ввода-вывода позволяют, то предпочтительнее 60-символьный алфавит, так как программы, составленные на нем, читаются легче. В настоящей книге используется 60-символьный алфавит. Полный список и сравнение этих двух наборов символов даны в приложении А.

Перфорирование операторов ПЛ/1 и данных. Исходная программа на языке ПЛ/1 может быть отперфорирована в колонках 2—72 перфокарт. Колонки 1 и 73—80 обычно транслятором не сканируются. В отличие от Фортрана одна карта в ПЛ/1 не образует запись. В конце каждого оператора ПЛ/1 стоит точка с запятой (;). Колонки 2—72 сканируются транслятором в непрерывном потоке. Один оператор от другого обязательно отделяется точкой с запятой. Для того чтобы сделать программу более легко читаемой, можно воспользоваться пробелами. Программист может отперфорировать операторы исходной програмы в непрерывном потоке, по одному на каждой карте или любым другим способом. В колонках 73—80 часто дается нумерация последовательности карт, но они могут служить и для других целей.

Как известно, пробелы помогают следить за программой. Но в некоторых местах пробелы не разрешаются (например, в сложных символах, идентификаторах, арифметических константах и в строках бит).

Комментарии рассматриваются как пробел, они могут быть помещеиы в любое место, где разрешеи пробел. Символы/* озиачают начало коммеитария, а */ — коиец комментария. В коммеитарии могут использоваться любые символы, иапример:

/* THIS ENTIRE COMMENT COULD BE PLACED ANYWHERE A BLANK COULD OCCUR*/ (этот законченный комментарий мог бы быть помещен всюду, где может быть пробел).

Возможность переработки инструкций и коммеитариев таким способом называется кодироваиием в свободной форме. Язык ПЛ/1 разрешает свободиое кодироваиие. Преимущества формата такого типа, если они еще не стали очевидиыми, вскоре стаиут ясиы.

Использоваиие колонок карт для перфорации операторов исходной программы на ПЛ/1. Колонка 1 не используется; колоики 2—72 — операторы ПЛ/1; колоики 73—80 не сканируются траислятором; оии могут служить для любой цели, ио обычио их используют для иумерации карт.

Все 80 колонок могут служить для перфорации входных и выходиых данных. Правильная организация даиных на перфокартах зависит от команд ввода-вывода в данной программе.

MAIN PROCEDURE (Главная процедура). Программа иа ПЛ/1 состоит из блоков операторов. Существуют два вида блоков: блок PROCEDURE и блок BEGIN (начало). Эти блоки могут быть вложены и/или траислироваться отдельио. Правила их обработки будут рассмотрены в следующей главе. Любая программа на ПЛ/1 обязателыио содержит блок PROCEDURE, который обычио определяют как главную PROCEDURE. Первый оператор в такой главиой PROCEDURE должен иметь вид:

```
label: PROCEDURE OPTIONS (MAIN);
```

где метка является идентификатором, состоящим из букв, цифр и служебных символов. Идентификатор может содержать от одного до семи символов, причем иачинаться он должен с буквы. (В подмиожестве ПЛ/1 метка ие может содержать более шести символов.)

Последний оператор PROCEDURE должеи иметь форму:

```
[label1:...] END [label2];
```

Метка 1 в этом операторе может ие использоваться (оиа будет описана в следующем разделе). Метка 2 тоже может не использоваться. Но в противиом случае это должна быть метка, данная PROCEDURE.

Операторы ПЛ/1 грубо можно разделить на две категории: операторы, содержащие информацию для транслятора, и операторы, которые выполияются в процессе решения. Последиие выполияются в порядке следования, если нет передачи управления на другой оператор. Оператор END, стоящий в конце главной процедуры, заканчивает выполнение программы. Операторы, содержащие информацию для транслятора, иеобходимы только во время траисляции, они не влияют на ход выполиеиия программы.

3.3. ИДЕНТИФИКАТОРЫ

Идентификатор — это один буквенно-цифровой символ или строка символов, причем первым символом должна быть обязательно буква алфавита. Как правило, идентификатор не может содержать более 31 символа. Идентификаторы могут быть именами или метками данных, файлов, операторов или точек входа различных частей программы. В некоторых особых ситуациях допустимая максимальная длина идентификатора может быть менее 31 символа. Как мы уже видели, одним из примеров меньшего количества символов может служить метка оператора PROCEDURE. Любой идентификатор из других внешних блоков может содержать не более семи символов. Такие идентификаторы называются внешними. (В подмножестве ПЛ/1 внешние операторы состоят из шести или менее символов.) Ключевые слова в ПЛ/1 тоже являются идентификаторами. Ключевым словом называется идентификатор, который имеет специальное значение для транслятора в соответствующем контексте. В главе 2 встречались ключевые слова

END, IF, WHILE, PUT, THEN, DO, ELSE и GET

Заметьте, что обозначение ключевых или резервированных слов в ПЛ/1 отличается от обозначения их в Фортране и Коболе. В ПЛ/1 ключевые слова имеют для транслятора специальное значение только в том случае, если они употреблены в соответствующем контексте. В 48-символьном алфавите некоторые слова всегда резервируются.

3.4. ТИПЫ ДАННЫХ

Данные в ПЛ/1 делятся на две основные категории: исходные данные и данные, управляющие программой. В свою очередь исходные данные делятся на два вида: арифметические, которые уже были рассмотрены, и строки символов. Исходные данные содержат информацию, которая должна быть обработана транслятором. Данные, управляющие программой, обеспечивают выполнение программы и состоят из метки (см. главу 2), события, задания, места и области данных. Сейчас будут рассмотрены не все эти типы данных.

Числа. При выборе чисел программист должен учитывать следующее:

основание может иметь форму BINARY (двоичный) или DECIMAL (десятичный);

масштаб может иметь форму FIXED (фиксированный) или FLOAT (плавающий);

тип может иметь форму REAL (вещественный) или COMPLEX (комплексный);

точность может быть ограничена возможностями вычислительной машины. (В подмножестве ПЛ/1 употребление константы COMPLEX запрещено. Все арифметические выражения полагаются REAL и поэтому описание REAL тоже не используется.)

Строка символов. Строка символов состоит из одного или нескольких последовательно расположенных символов, заключенных в кавычки. Строка символов рассматривается как единица данных. Она может содержать любой символ, допускаемый данной вычислительной машиной. Длиной символьной строки называется количество символов, заключенных в кавычки. (Заметьте, что пробел в ПЛ/1 — тоже символ, он должен учитываться как часть символьной строки.)

1. Если в символьную строку включен комментарий, то и комментарий и его ограничители (/* и */) учитываются при определении длины строки.

2. Если в строке символов появляется апостроф, то его заменяют двумя апострофами ('') без пробелов между ними. Два апострофа в этом случае принимаются за один символ. Кавычки имеют форму двух апострофов.

На каждый символ в строке в памяти машины IBM-360 отводится один байт. Максимальная длина символьной строки в этой системе 32 767 символов. (В подмножестве ПЛ/1 максимальная длина равна 255.) Перед символьной строкой может стоять коэффициент повторения. Он должен быть целым числом без знака и записывается в скобках перед строкой.

Далее приведены примеры констант символьных строк:

1. 'THIS IS AN ACCEPTABLE CHARACTER STRING'' (длина = 38)
2. '3X — 4Y = 8' (длина = 7)
3. 'DOLLARS — > MARKS' (длина = 16)
4. 'BEETHOVEN''S''''NINTH SYMPHONY''''' (длина = 30)
5. (3) 'ABC' (длина = 9)
6. 'AB /* COMMENT*/ C' (длина = 3)
7. 'CHARGOGGAGOGGMANCHAUGAGOGGCHAUBUNA GUNGAMAUGG' (длина = 44)
8. '1G1 K2+1 MC 20H 1Q 2QJ/2 (20 19) 18Q 5/4Q · 3E 2H/' (длина = 54)

В примерах 4, 5, 6 приводятся константы BEETHOVEN'S "NINTH SYMPHONY" (девятая симфония Бетховена), ABCABCABC, ABC соответственно. В примере 7 дается название озера в штате Массачусетс. В примере 8 приведена музыкальная фраза.

Строка бит. Строка бит представляет собой непрерывную последовательность двоичных цифр, заключенных в кавычки. После закрывающей кавычки пишется буква B. Длина строки бит определяется числом бит, заключенных в кавычки. В машине IBM-360 строки бит хранятся по восемь бит на байт. Максимальная длина строки бит 32 767. (В подмножестве ПЛ/1 максимальная длина строки бит 64.) Перед строкой бит может стоять коэффициент повторения. Он записывается перед строкой в круглых скобках и является целым числом без знака. Приведем примеры строк бит:

'110110110'B (длина = 9. Может быть записана как (3)'110'B)
'1'B (длина = 1)
(10)'0'B (длина = 10. Может быть записана как '0000000000'B)

Метки. Единицей данных тнпа меткн служит метка в виде константы или значение переменной типа метки. Метка в виде константы — это идентификатор (определение идентификатора дано в параграфе 3.3), который нспльзуется как приставка к оператору ПЛ/1, что прнводит во время выполнения программы к передаче управления на этот оператор. Все метки, которые были приведены в главе 2, были константамн. За константой типа метки всегда стоит двоеточие (:), отделяющее метку от оператора, к которому она относнтся. Переменные типа метки будут рассмотрены в следующем параграфе.

3.5. ОПЕРАТОР DECLARE (ОБЪЯВИТЬ) И DEFAULT (УМОЛЧАТЬ)

Многомерные массивы. В главе 2 было показано, что оператор DECLARE в примере DECLARE A (5, 10) REAL DECIMAL FIXED (5); объявляет, что А представляет собой массив 50 вещественных десятичных фнксированных чисел с точностью 5. Напомним, что информация о размерности должна следовать сразу же за ндентификатором переменной с пробелом или без него. Массивы расположены в памяти машины таким образом: чем правее индекс, тем быстрее он изменяется. Допускается не более 32 индексов. (В подмножестве ПЛ/1 можно использовать не более 3 нндексов.)

В рассмотренном примере массив А будет размещен в оперативной памяти в следующем порядке:

```
A (1,1), A (1,2),..., A(1,10), A (2,1), A (2,2),..., A (2,10), A (3,1),
A (3,2),..., A (3,10), A (4,1), A (4,2), ..., A (4,10), A (5,1), A (5,2),...,
A (5,10)
```

Транслятор ПЛ/1 считает нижний индекс массива равным единице. Однако программист может указывать (если это необходимо) как верхнюю, так и нижнюю границу массива. (В подмножестве ПЛ/1 нижняя граница массива не задается. Она всегда полагается равной 1.) A (—2 : 10,0 : 10) объявляет, что А — это массив размерностью 13 × 11, причем первый индекс изменяется от—2 до 10 с приращением 1, а второй — от 0 до 10 также с приращением 1.

Верхняя и ннжняя границы индекса разделены двоеточием, а сами индексы отделены друг от друга запятой. По сравнению с Фортраном такой порядок имеет несомненное преимущество, так как очень часто алгоритм гораздо удобнее запрограммировать с помощью нулевого илн отрицательного индекса.

Начальные значения. (В подмножестве ПЛ/1 употребление описателя INITIAL запрещено.) Начальные значения переменной могут быть объявлены оператором DECLARE с записью опнсателя INITIAL. Например,

```
DECLARE A REAL FIXED DECIMAL (5) INITIAL (25);
```

Это означает, что переменной А присваивается значенне 00025.

Перед начальным значением в круглых скобках может стоять коэффициент повторения, как это показано в следующем примере:

```
DECLARE A (20) REAL FIXED DECIMAL (5) INITIAL (1, 2, (18)0);
```

где A (1) присвоено значение 00001, A (2) — 00002, а переменным A (3) — A (20) — значение 00000.

Начальные значения массиву размерностью 3 × 5 присваиваются с помощью оператора DECLARE. В первом столбце будут располагаться единицы, во всех остальных — нули:

```
1 0 0 0 0
1 0 0 0 0
1 0 0 0 0
```

Следует обратить внимание на использование вложенных коэффициентов повторения:

```
DECLARE A(3,5) REAL FIXED DECIMAL (1) INITIAL ((3) (1, (4) 0));
```

Интересным примером с вложенными повторителями является присваивание начальных значений единичной матрице, которая представляет собой квадратный массив, все элементы которого, расположенные на главной диагонали, равны единице, а остальные — нулю. Этот метод показан на примере массива рзмерностью 20 × 20.

```
DECLARE A (20, 20), REAL FIXED DECIMAL (5) INITIAL (1, (19) ((20) 0,1));
```

В некоторых случаях программисту бывает необходимо присвоить начальные значения только части массива без указания остальных начальных значений. Если в операторе INITIAL стоит звездочка (*), то это означает, что соответствующее значение переменной не задается. В следующем примере переменным A(—2), A (0), A (2), A (4) присваиваются начальные значения, равные единице, а переменным A (—1), A (1), A (3), A (5) не присваивается никаких начальных значений:

```
DECLARE A (—2 : 5) REAL FIXED DECIMAL (5) INITIAL ((4) (1,*));
```

В главе 2 в операторе DECLARE давались характеристики каждого описателя идентификатора. Это очень утомительно и требует много времени. Мы уже рассмотрели один способ упрощения подобной задачи путем употребления повторителей в тех случаях, когда идентификаторы имеют общие описатели. Это показано в следующем примере:

```
DECLARE (A FIXED (6,2), B (8) FLOAT (5)) REAL DECIMAL;
```

Можно также воспользоваться сокращениями. Транслятор распознает сокращения нескольких длинных ключевых слов, употребляемых в ПЛ/1. Полный список таких слов приведен в приложении. Далее перечислены те ключевые слова, которые уже встречались в этой книге (они будут введены и в настоящей главе):

PROCEDURE	PROC
DECLARE	DCL
BINARY	BIN
DECIMAL	DEC
CHARACTER	CHAR
COMPLEX	CPLX
INITIAL	INIT
VARYING	VAR

(В подмножестве ПЛ/1 сокращения не разрешаются.) Сокращения могут употребляться в операторах ПЛ/1 всякий раз, когда имеются ключевые слова.

Оператор

```
DCL (A, B (5,—1 : 6)) REAL DEC FLOAT (6);
```

эквивалентен

```
DECLARE (A, B (5,—1 : 6)) REAL DECIMAL FLOAT (6);
```

Еще одним способом упрощения записи оператора DECLARE может быть система умолчания. Если описатель идентификатора не определен оператором DECLARE, то транслятор определит пропущенные описатели по контексту, в котором употреблен этот идентификатор. Это делается с помощью специального набора правил, которые выполняются в зависимости от ситуации. Правила работы по умолчанию будут рассмотрены в следующих параграфах при рассмотрении соответствующих типов данных. Точность вычислений при работе по умолчанию (как и в некоторых других случаях) в значительной степени зависит от вычислительной машины. В настоящей книге рассматривается работа по умолчанию для машины IBM-360.

Описатели идентификаторов могут определяться программистом различными способами. Если все описатели задаются в операторе DECLARE, то такой способ называется *явным объявлением*; если не определен ни один из описателей, то способ называется *неявным объявлением*; если несколько (но не все) описатели заданы, то это *частично явное объявление*. В главе 2 почти все арифметические идентификаторы были объявлены явно.

В следующей таблице приведены описатели данных, которые будут рассматриваться в следующих параграфах.

ОПИСАТЕЛИ ДАННЫХ

Данные	Подкласс	Возможные описатели
Число	Основание	DECIMAL или BINARY (десятичное или двоичное число)
	Масштаб	FLOAT или FIXED (плавающее или фиксированное)
	Тип	REAL или COMPLEX (вещественное или комплексное)
	Точность	(W [,d])
		INITIAL (начальные значения)
Строка	Бит	BIT (длина)
		INITIAL (начальные значения)
		VARYING (переменная длина)
	Символ	CHARACTER (длина)
		INITIAL (начальные значения)
		VARYING (переменная длина)
Метка		LABEL [(значения метки)]

Идентификаторы в арифметических выражениях. (В подмножестве ПЛ/1 все переменные в арифметических выражениях полагаются REAL, поэтому описатели COMPLEX и REAL не разрешаются.) Если нет явного объявления типа переменной, то полагают, что пере-

менная представляет собой число. Если, кроме этого, переменная не описана явно, то первый символ определяет ее следующим образом:

а) если первый символ буквы I, J, K, L, M или N, то это переменная типа BINARY FIXED REAL (15);

б) если первый символ — любая буква алфавита кроме I, J, K, L, M или N, то это переменная DECIMAL FLOAT REAL (6).

Обратите внимание на то, что рассмотренное правило применяется только тогда, когда ни основания, ни масштаб, ни тип явно не объявляются. Программист должен быть осторожен с необъявленными явно переменными, идентификаторы которых начинаются с букв I, J, K, L, M или N. Эти переменные представляют собой двоичные целые числа. Операции с такими числами нужно производить очень внимательно: если в них появляются дробные части, то они будут отброшены так же, как это делается в Фортране.

При частично явном объявлении арифметических идентификаторов необходимо выполнять следующие правила:

а) если основание явно не объявлено, то переменная неявно объявляется DECIMAL по умолчанию;

б) если масштаб явно не объявлен, то он неявно объявляется FLOAT по умолчанию;

в) если тип явно не объявлен, то он неявно объявляется REAL по умолчанию;

г) если точность явно не объявлена, то в зависимости от типа электронной вычислительной машины она устанавливается по умолчанию. Для машины IBM-360 точность объявляется следующим образом:

```
DECIMAL FIXED (5,0)
BINARY FIXED (15,0)
DECIMAL FLOAT (6)
BINARY FLOAT (21)
```

Далее приведена таблица описателей данных по умолчанию.

ОПИСАТЕЛИ ПО УМОЛЧАНИЮ ДЛЯ ИДЕНТИФИКАТОРОВ В АРИФМЕТИЧЕСКИХ ВЫРАЖЕНИЯХ

	Неявное объявление		Частично явное объявление
Подкласс	Первые символы I, J, K, L M или N	Первые символы не I, J, K, L, M или N	Необъявленные описатели
Основание	BINARY	DECIMAL	DECIMAL
Масштаб	FIXED	FLOAT	FLOAT
Тип	REAL	REAL	REAL

Числа с фиксированной точкой являются правоустановленными и, если это необходимо, левая часть поля будет заполнена нулями.

Строки данных — левоустановленные и в случае символьных строк правая сторона будет заполнена пробелами, а в случае строк битов — нулями. Если строка слишком длинна, то отбрасывание

ТОЧНОСТЬ ОПИСЫВАЕМЫХ ДАННЫХ ПО УМОЛЧАНИЮ ДЛЯ СИСТЕМЫ IBM-360 И МАКСИМАЛЬНАЯ ТОЧНОСТЬ ДЛЯЧИСЕЛ И СТРОК

Тип переменной	Точность даиных по умолчанию для языка ПЛ/I и его подмножеств	Максимальная точиость для ПЛ/1	Максимальная точность для подмножества ПЛ/1
FIXED BINARY	(15,0)	31 бнт	31 бнт
FIXED DECIMAL	(5,0)	15 цифр	15 цнфр
FLOAT BINARY	(21)	53 бита	53 бита
FLOAT DECIMAL	(6)	16 цифр	16 цифр
CHARACTER	неявное объявленне запрещено; длииа должна быть определена	от 0 до 32 767 снмволов	от 1 до 255 снмволов
BIT	неявное объявлеине запрещено; длииа должна быть определена	от 0 до 32 767 битов	от 1 до 64 бнтов

лишних символов производится справа. Прн отбрасыванни сообщення об ошибке не поступает.

Следующне примеры содержат методы объявления арнфметнческнх переменных.

а) DCL A (10), (B, I) DECIMAL (10);

В этом примере переменная A представляет собой одномерный массив размерностью 10 с описателями DECIMAL, FLOAT, REAL, каждый элемент массива имеет точность (6). B и I — DECIMAL, FLOAT, REAL с точностью (10).

б) Идентнфнкаторы ICOUNT, X н Y не появляются в операторе DECLARE, но в программе могут служить арифметическнмн переменными. ICOUNT по умолчанню объявляется REAL BINARY FLOAT (15). X и Y по умолчанню объявляются REAL DECIMAL FLOAT (6).

в) DCL 1 (5,10), X (10) INIT ((10) 0), A BIN (20), N FIXED; где I имеет размерность 5 на 10 н является BINARYFIXED REAL (15). X имеет размерность 10 и является DECIMAL FLOAT REAL (6) н каждой переменной X присваивается начальное значение 0. Переменной A приписываются описатели BINARY FLOAT REAL (20). N — DECIMAL FIXED REAL (5).

Символьные строки и строки битов. Идентнфнкаторы для хранения символов н битов строк в памяти машины объявляются описателями CHARACTER н BIT; за ннмн в круглых скобках указывается длина строкн. Начальные значення могут задаваться опнсателем INITIAL, как для арифметическнх переменных, с той только разннцей, что строка в отличие от арифметической константы должна быть определена.

Примеры

Оператор DCL A CHARACTER (20); объявляет A символьной строкой с длиной в 20 символов.

Оператор DCL A (10) BIT (20); объявляет A индексированным массивом размерностью 10 и каждый нндекс определяет строку битов длиной 20.

Оператор DCL (A INITIAL ('ABCDE'), B INITIAL ('FG')) CHAR (5); объявляет, что как A, так и B — строки символов длиной 5. Символы ABCDE присваиваются переменной A, а переменной B присваиваются символы FG, за которыми следуют три пропуска.

Оператор DCL A BIT (8) INIT ('1100'B); объявляет переменную A, равной строке битов длиной 8, и ей присваивается начальное значение 1100 битов, за которым следуют четыре нуля.

Строки символов и битов могут быть также объявлены переменной длины. В этом случае добавляется описатель VARYING. Максимальная длина строки должна быть определена, а текущая длина строки равна числу символов, хранящихся в памяти машины в данный момент. (В подмножестве ПЛ/1 употребление описателя VARYING запрещено.) Например, оператор DECLARE LAST_NAME (50) CHARACTER (20) VARYING; объявляет, что идентификатор LAST_NAME является именем массива из 50 строк, а каждая строка символов имеет максимальную длину 20. Если во время выполнения программы строка символов 'JONES' хранится в LAST_NAME (7), то длина этой строки символов будет равна пяти.

Оператор DCL (B BIT (20) INIT ((10)'1'B), A CHAR (6)) VAR; объявляет, что B является строкой битов переменной длины; ее максимальная длина 20. В данном примере длина строки равна 10 и в ней хранится 10 единичек. A есть строка символов с переменной длиной; ее максимальная длина 6.

Метки. Все метки, встречавшиеся в главе 2, были метками-константами. Их употребление аналогично употреблению меток в Фортране. Но в ПЛ/1, кроме чисел, в качестве меток можно использовать имена. В ПЛ/1 метка может быть объявлена переменной и ее значение во время выполнения программы будет меняться в процессе выполнения программы в зависимости от описателя LABEL. Это более общий подход по сравнению с Фортраном. Метки могут меняться во время выполнения программы, служить параметрами при обращении к подпрограммам и т. д. Например, оператор DECLARE INPUT LABEL; объявляет, что идентификатор INPUT представляет собой метку-переменную. Рассмотрим простой пример с меткой-переменной:

```
          DECLARE XYZ LABEL;
                    .
                    .
                    .
ABC : выражение;
                    .
                    .
                    .
D4 ; выражение;
                    .
                    .
                    .
          XYZ=ABC;
                    .
                    .
                    .
          GO TO XYZ;
                    .
                    .
                    .
```

ABC и D4 представляют собой метки-константы, а XYZ — метка-переменная.

Рассмотрим другой пример:

```
DECLARE LABEL___X (5) LABEL INITIAL (A, B, C, D, E);
```

Этот оператор объявляет LABEL_X массивом, состоящим из пяти меток-переменных, и приписывает им значения меток-констант A, B, C, D и E соответственно. Если во время выполнения программы встречается оператор GO TO LABEL_X (I), то происходит передача управления на операторы с метками A, B, C, D или E в зависимости от значения I.

Для оптимизации рабочей программы описатель LABEL может также включать список всех меток-констант, которые могут присваиваться меткам-переменным во время выполнения программы. Такой список дается сразу же за описателем LABEL и заключается в круглые скобки. Метки-константы отделены друг от друга запятыми. Если такой список задан, то во время выполнения программы метке-переменной может присваиваться только одно из значений констант, имеющихся в списке. (В подмножестве ПЛ/1 употребление меток-переменных не допускается.)

Пример

```
DECLARE XYZ LABEL (ABC,D4);
```

3.6. УПРАЖНЕНИЯ

Короткие упражнения

1. В следующем операторе DECLARE определите подразумеваемые по умолчанию описатели для каждого идентификатора в арифметических выражениях:

```
DECLARE A FIXED (6,2),
        B FLOAT (5),
        C (5),
        D BINARY,
        E DECIMAL,
        F REAL DECIMAL,
        G REAL BINARY DECIMAL,
        H REAL FLOAT DECIMAL,
        I (16),
        J DECIMAL;
```

2. Напишите описатель INITIAL для массива A (15, 15), если:

а) каждый элемент равен 0;

б) все элементы на главной диагонали не определены, а остальные равны 1;

в) элементы второго столбца равны 1, а элементы пятого равны 2; все другие элементы равны 0.

3. Если приведенный оператор DECLARE представляет собой часть программы на ПЛ/1, то в каком порядке будут храниться в памяти машины начальные значения?

```
DCL (A INIT ('X'), B INIT ('A''BC'), INIT ('THIS IS A CHARACTER STRING CONSTANT')
('это константа, состоящая из строки символов') CHAR (10),
(X INIT ((4)'10'B), Y INIT ('101'B)) BIT (8),
(I INIT (32), J INIT (54321)) FIXED (4),
K INIT (101101B),
Z FIXED INIT (5.26);
```

4. Оператор GO TO в Фортране имеет форму GO TO (s_1, s_2, ..., s_n), I, где каждый элемент s_i — число, и если I = к, то управление передается на оператор $s_к$. Напишите на ПЛ/1 эквивалент следующего оператора Фортрана:

```
GO TO (23, 14, 62, 4, 7), I
```

Задачи для программирования

1. Существуют много способов вычисления значений функций методом последовательной аппроксимации. Напишите программу для вычисления и печатания значений квадратных корней положительных одно-, двух-, трех-, четырех и пятизначных целых чисел. Для вычисления квадратных корней воспользуйтесь следующим алгоритмом, основанным на методе последовательной аппроксимации Ньютона:

а) считать число a;

б) если число a меньше 5, то первым приближением будет число 2, иначе говоря, $a/2$;

в) значения последовательных приближений X_{n+1} вычисляются по рекуррентной формуле

$$X_{n+1} = \frac{X_n^2 + a}{2X_n},$$

где X_n — текущее приближение; a — исходное число;

г) продолжайте вычисления до тех пор, пока модуль выражения $(X_{n+1})^2 - a$ не станет меньше 0,0001.

Пример

$a = 9$, $a > 5$. Следовательно, $X_1 = 4.5$.

Первый шаг: $X_2 = \frac{X_1^2 + a}{2X_1} + \frac{(4.5)^2 + 9}{2(4.5)} + \frac{13}{4} = 3.25$

$(3.25)^2 - 9 > 0.0001.$

Второй шаг: $X_3 = \frac{X_2^2 + a}{2X_2} = \frac{(3.25)^2 + 9}{2(3.25)} 3.0096$

$(3.0096)^2 - 9 > 0.0001.$

Третий шаг: $X_4 = \frac{X_3^2 + a}{2X_3}$ и т. д.

Вычисления на каждом шаге выполняются очень быстро. Однако существует опасность, что программа будет выполняться бесконечно. В подобных ситуациях необходимо очень тщательно следить за тем, чтобы переменные объявлялись с удовлетворительной точностью так, чтобы можно было провести проверку на окончание цикла.

2. В предыдущей главе была показана сортировка чисел методом перестановки. Другой метод, называемый методом сортировки чисел по основанию системы счисления, предполагает выделение отдельного элемента памяти для каждой цифры, а затем группировку чисел по цифрам в каждом последовательном проходе.

Напишите программу для сортировки чисел по основанию системы счисления для 50 двузначных чисел и напечатайте результаты. Такая программа потребует 10 участков памяти для 50 двузначных целых чисел при первом проходе (для сортировки единиц) и еще 10 — при втором проходе (для сортировки десятков).

После проведения этих операций выделенная память может быть снова использоваиа.

В то время как даиные считываются, оии посылаются в разлнчиые участки памяти в зависимости от значения цифр в разряде едииии. Прн следующем проходе даиные считываются иачиная с иуля в разряде единиц, а затем посылаются в соответствующие участки памяти в разряды десятков. Затем даиные считываются последовательиио с самых иизших позиций (в последующем примере — самые правые), отсортироваиный список чнсел печатается в восходящем порядке. Следующий пример иллюстрирует этот алгоритм.

Входные даиные	23,			45,		11,		14,		18,		61,		83,		02,		00,		99,	49,	76
Проход едииц	0	1	2	3		4		5		6	7	8		9								
	00	11	02	23		14		45		76		18		99								
		61		83										49								
Запись чисел после первого прохода	00,			11,		61,		02,		23,		83,		14,		45,		76,		18,	99,	49
Проход десятков	0			1		2		3		4		5		6		7		8		9		
	00			11		23				45				61		76		83		99		
	02			14						49												
				18																		
Окончательиый результат	00,			02,		11,		14,		18,		23,		45,		49,		61,		76,	83,	99

Такой вид алгоритма используется в устройствах для механической сортировки перфокарт.

3. Еще один способ сортировкн—«магазнниая» сортировка. В этом алгоритме данные считываются и проверяются по иачальной последовательности. Если такая последовательиость существует, то даиные заиосятся в список последовательиости, в противиом случае даиные направляются в рабочий список. В конце такого прохода входиых данных все даиные рабочего списка сравиваются со списком последовательиостн и в том случае, когда для числа нз рабочего списка найдено место в последовательиостн, остаток последовательиостн сдвигается, для того чтобы освободить место для иового члеиа.

Напишнте программу для сортировки входиых даиных (до 500 едииии), представляющих собой целые числа, состоящие ие более чем из трех цифр, с помощью метода «магазиниой» сортнровки и иапечатайте спнсок этих чисел. Далее дается подробиое описаиие приведениого раиее алгоритма:

а) устаиовите две области хранеиия даиных, каждая из которых содержнт 500 ячеек. Одна иазывается списком последовательности, вторая — рабочим списком;

б) считайте первое чнсло в спнсок последовательиостн;

в) считайте последовательиый иабор чисел: сравнните каждое чнсло с самым большнм числом списка последовательности. Еслн какое-либо число превосходит его или равио ему, введите это число в список последовательности и используйте его как новое самое большое число. Оргаиизуйте счетчики как для счета всех считаиных чисел, так и для чисел, записаиных в список последовательности. Когда количество чисел в списке последовательиости станет равиым общему колнчеству считываемых даиных, сортировка закаичивается;

г) если входное число меиьше иаибольшего в списке последовательиости, заиесите его в рабочий список;

д) когда все входиые даиные считаиы, а в рабочем списке еще имеются числа, каждое число рабочего списка последовательио сравиивайте с числами списка последовательиостн до тех пор, пока в списке последовательиости ие будет найдеио число, большее чнсла в рабочем списке. Затем сдвниьте все числа в списке последовательиости винз, а иайденное число рабочего списка заиесите на осво-

боднвшееся место. Увелнчьте счетчнк числа элемеитов в списке последовательиости иа едиицу н проверьте, аакоичеиа лн сортнровка.

4. Напишите программу для вычнсления зиачення e — осиования иатуральиых логарифмов — до 10 значащих цнфр, нспользуя следующнй ряд:

$$e = 1 + \frac{1}{1} + \frac{1}{1\cdot 2} + \frac{1}{1\cdot 2\cdot 3} + \frac{1}{1\cdot 2\cdot 3\cdot 4} + \ldots + \frac{1}{1\cdot 2\cdot 3\ldots n}.$$

Группа $1\cdot 2\cdot 3\cdot 4\cdot\ldots\cdot n$ иазывается n-факторнал и сокращенио пишется $n!$

Для иахождеиия требуемой точиости добавьте столько члеиов, сколько иеобходимо. Выдайте иа печать число e и колнчество нспользоваиных члеиов ряда.

5. Условиая вероятиость определяется как отиошеине числа событий, которые произошлн, к общему числу возможиых событнй. Напишнте программу, которая подсчитает н иапечатает условиые вероятности получеиия чнсел от 2 до 12 прн бросаннн двух шестиграиных нгральиых костей. Прнмеинте следующнй алгоритм:

а) поскольку нмеются две шестнграиные костн, то существует 6×6 (т. е. 36) возможных случаев. Это чнсло н будет зиаменателем;

б) возможиые события — выпадеиие чнсел 2, 3, 4, ..., 12. Этн чнсла (числители) геиернруются в результате бросаиия каждой кости по очередн, т. е. 1 иа первой кости при выпадеинн 1, 2, 3, ..., 6 иа второй; подсчет пронзводнтся по всем событиям 2, 3, 4, ..., 12; затем выпадеиие 2 иа первой костн прн выпадеиии 1, 2, 3, ..., 6 иа второй. И сиова подсчет пронзводится по всем событиям. Бросаине продолжают до тех пор, пока иа первой кости ие выпадет цифра 6.

Этот метод можио применять для вычнслеиня условиых вероятиостей во миогих вндах нгр н реальных снтуациях.

6. В четырех соседних городах Нордфнлде, Саусфилде, Истфнлде и Вестфилде иалогн иа продажу товаров установлеиы в размерах 3,3,5,4 н 2% соответственно. На картах отперфорнроваио сиачала иазваиие одиого нз городов в внде строкн снмволов (заключеио в кавычки), затем следует по крайией мере однн пробел, а затем отперфорирована сумма товара (в долларах и цеитах). Напишите программу, которая будет считывать пронзвольиое чнсло такнх перфокарт и выдавать иа печать иазванне города, цеиу товара, велнчниу иалога и общую стонмость (товар плюс иалог). Можио считать, что ин одии товар не продается по цеие выше 9999 долларов 99 цеитов.

Глава 4

ВЫРАЖЕНИЯ, ПРИСВАИВАНИЕ ЗНАЧЕНИЙ И УПРАВЛЕНИЕ ПРОГРАММОЙ

4.1. ВВЕДЕНИЕ

В главе 2 были рассмотрены основные понятия и правила образования выражений и обозначения операторов присваивания и управления программой. В настоящей главе эти правила будут обсуждены более детально, а позже будут рассмотрены и дополнительные типы данных.

В большинстве языков программирования высокого уровня выражения являются категорией значения. Они могут состоять из константы или переменной, комбинации констант, переменных, символов операций и круглых скобок. В ПЛ/1 существуют три различных типа выражений: переменные, массивы ис труктуры. Выражения типа переменной, как показывает название, определяют значение переменной. Почти все выражения, встречавшиеся в настоящей книге до сих пор, представляют собой выражения этого типа. Выражение типа массива определяет значение массива. Единственными выражениями этого типа, с которыми мы уже встречались, являются выражения GET или PUT LIST (A), где A — идентификатор массива. В Фортране это единственное допустимое выражение для обозначения массива. В ПЛ/1 возможности употребления такого рода выражений значительно расширены. Структуры или выражения типа структур до сих пор не встречались. Они будут рассмотрены в главе 7.

В ПЛ/1 нет специального ключевого слова для обозначения присваивания. Символ присваивания (=), знак равенства, показывает, что значение выражения справа от символа присваивается переменной, находящейся слева от этого символа.

В главе 2 были введены все операторы ПЛ/1, кроме одного оператора строки символов ||, означающего конкатенацию*. Этот опе-

* Конкатенация значит сцепление, соединение. — *Примеч. пер.*

ратор будет рассматриваться в параграфе 4.3. Приоритеты выполнения всех этих операций, которые показывают порядок выполнения выражений, даются в следующей таблице:

Символ	Тип	Значение	Приоритет
**	Арифметический	Возведение в степень	Самый высокий
¬	Строка битов	Символ отрицания	Самый высокий
+	Арифметический	Префикс +	Самый высокий
−	Арифметический	Префикс —	Самый высокий
/	Арифметический	Знак деления	Второй
*	Арифметический	Знак умножения	Второй
+	Арифметический	Инфикс +	Третий
−	Арифметический	Инфикс —	Третий
||	Строка	Конкатенация	Четвертый
>=	Сравнение	Больше или равно	Пятый
<=	Сравнение	Меньше или равно	Пятый
>	Сравнение	Больше	Пятый
<	Сравнение	Меньше	Пятый
¬=	Сравнение	Не равно	Пятый
=	Сравнение	Равно	Пятый
¬>	Сравнение	Не больше	Пятый
¬<	Сравнение	Не меньше	Пятый
&	Строка битов	И	Шестой
|	Строка битов	Или	Самый нижний

Внутри каждого уровня операторы имеют равный приоритет. Они выполняются в порядке появления в каждом данном уровне: слева направо для уровней от 2 до 7 и справа налево для самого высокого приоритета. Этот порядок может быть изменен с помощью круглых скобок. Выражения, заключенные в круглые скобки, начинают выполняться с тех, которые заключены в самые внутренние скобки.

Замечание. Знак равенства, кроме употребления в качестве оператора сравнения, может употребляться как оператор присваивания.

При написании выражений на языке ПЛ/1 программист может воспользоваться круглыми скобками для изменения порядка выполнения операций по приоритетам, чтобы получить желаемые результаты. Так, выражение (A + B)/D не аналогично A + B/D. С помощью круглых скобок можно также упростить чтение сложных выражений, если скобки не требуются для других обозначений. Начинающему программисту это очень облегчает задачу, однако введение лишних скобок увеличивает продолжительность трансляции. Во время трансляции выражения, не содержащего круглых скобок, транслятор сначала сканирует выражение справа налево, находя операторы самого высокого приоритета, а затем — слева направо, находя операторы второго уровня, и т. д. Таким образом, выражение сканируется семь раз*. Если в транслируемом выражении имеются круглые

* Существуют более эффективные алгоритмы трансляции, не требующие семикратного сканирования. — *Примеч. пер.*

скобкн, то эти семь скаииронваний пронзводятся отдельио для каждого элемеита, заключеииого в скобки, начниая с самых внутренних скобок до самых внешних, до тех пор, пока не будет протранслнровано все выражеине.

В выражениях могут быть также употреблеиы идеитификаторы для обозиачения даииых смешанных типов. Эти различиые типы данных, еслн иеобходимо, в процессе траисляцин должны быть преобразоваиы в соответствнн с приоритетом операций. Обычно программисту нет иадобности уделять этому особое внимаине, так как преобразоваиие в тип самого высокого уровия пронзводится только при крайией необходимости, а задаиная точиость всегда обеспечивается. Преобразоваиие типов данных может быть также вызваио выполненнем самого оператора присваивания. Если идеитификатор левой частн выражеиня не прииадлежит к тому же типу, что и выраженне в правой частн, то после вычислеиия выраженя результат должен быть преобразован к типу ндентнфикатора. Следовательио, даже простой оператор ПЛ/1 может потребовать выполиения гораздо большего числа шагов, чем это может показаться прн беглом рассмотреиии. Перед выполнением какой-лнбо одной операцни даиные могут преобразовываться для полученя промежуточиого результата. Эти результаты, в свою очередь, могут быть преобразованы еще раз до соединения их с другим элементом даииых в следующей операцнн и т.д. Еслн конечиый результат выражен не тем типом даиных, который требует идеитнфнкатор, стоящий в операторе присваиваиия слева от разделительного снмвола, может потребоваться окоичательиое преобразоваиие тнпа даиых. По сравиению с Фортраиом, где смешаиные тнпы даиых либо абсолютио запрещены, либо в лучшем случае их употреблеиие очень ограничено, ПЛ/1 дает большне преимущества*. Преобразоваиие даиых в выраженин в тех случаях, когда в этом нет иеобходимости, увеличнвает время выполнеиия программы, ио иа это ие стонт обращать виимаия на ранних стаднях обучения программированию. Примеры преобразоваиия разлнчиых типов даиых будут прнведены в следующих параграфах настоящей главы.

4.2. ВЫРАЖЕНИЯ И ОПЕРАТОРЫ ПРИСВАИВАНИЯ, СОДЕРЖАЩИЕ АРИФМЕТИЧЕСКИЕ ДАННЫЕ

В главе 2 уже рассматрнвалнсь выражения и операторы присванваиня, содержащие арифметические даиные. Правила преобразоваиия арифметических даиных могут быть обобщеиы следующим образом.

Каждая операция является либо одиоместной (одии операид), либо двухместиой (два операида). Эти операиды могут быть констаитамн, отдельиыми переменными или иекоторым подвыражением.

---

* В Фортране предусмотрены достаточно широкие возможности работы со смешанными типами данных. — *Примеч. пер.*

Каждый операнд может иметь описатели: десятичный или двоичный, фиксированный или с плавающей точкой, вещественный или комплексный, а также иметь определенную точность. Если один операнд представляет собой константу, то он обычно преобразовывается в процессе трансляции в соответствии с описателями другого операнда. Если оба операнда либо константы, либо переменные с различными описателями, то один из операндов будет преобразован соответствующим способом:

а) если один операнд фиксированный, а другой с плавающей точкой, то фиксированный операнд должен быть преобразован в операнд с плавающей точкой, результат будет выдан в формате с плавающей точкой;

б) если один операнд десятичный, а другой двоичный, то десятичный операнд преобразуется в двоичный, результат выдается в двоичной форме;

в) если один операнд вещественный, а другой комплексный, то никаких преобразований не производится, в результате получается комплексное число;

г) точность промежуточного результата сохраняется.

Правила преобразования данных зависят от внутреннего представления элементов данных в соответствующем устройстве. Обычно преобразование производится так, чтобы получить в результате максимальную точность. В случае необходимости можно воспользоваться справочником по ПЛ/1 для устройства, на котором выполняется программа. Для иллюстрации того, как делаются преобразования, рассмотрим следующий пример:

```
DCL A REAL FLOAT DECIMAL (4) INITIAL (2),
    B REAL FIXED DECIMAL (2) INITIAL (3),
    C REAL FIXED DECIMAL (4,2) INITIAL (4),
    Z COMPLEX FIXED DECIMAL (5,3);
Z = A* (B+C)+2;
```

Начальные значения A, B и C запишем в память следующим образом:

```
A       2000E01
B       03
C       04.00
```

Первая операция, которую необходимо выполнить, будет B + C. Оба операнда — вещественные фиксированные десятичные числа, но с различной точностью. При выполнении этого сложения существует возможность «переноса» в разряд сотен.

Следовательно, в результате получится вещественное фиксированное десятичное число с точностью (5, 2).

Второй операцией будет A* (B + C). Оба операнда — вещественные десятичные числа, но один из них — фиксированный, а второй — с плавающей точкой. Промежуточный результат 007.00 преобразуется в число с плавающей точкой с сохранением всех цифр, а затем выполняется операция A* (B + C). В результате получается вещественное десятичное число с плавающей точкой с точностью 5. Результат будет .14000E02.

В третьей операции к этому числу добавляется константа 2, что дает в результате .16000E02. Последним шагом при выполнении этого оператора будет присваивание переменной значения, полученного при выполнении третьей операции. Но переменная Z по сравнению с полученным числом имеет другие описатели. Следовательно, требуется произвести еще одно преобразование. Число 16.000+00.0001 будет храниться в Z.

В ПЛ/1 гораздо больше встроенных функций, чем в других языках программирования. Таблица этих функций дана в приложении Д. Аргументы большинства таких функций не ограничены одним типом данных, как это делается в Фортране, а полученное значение функции обладает описателями, которые определяются типом функции и аргументами. Например, функция SQRT используется в Фортране для получения квадратного корня от числа с плавающей точкой и одинарной точностью, функция DSQRT — от числа с плавающей точкой и удвоенной точностью, функция CSQRT — от комплексного числа с одинарной точностью, функция CDSQRT — от комплексного числа с удвоенной точностью.

В ПЛ/1 функция SQRT применима для всех арифметических аргументов, даже для имен массивов, как это будет показано далее в этой главе. Кроме того, аргументы функций сами могут быть выражениями.

Если в выражении встречается функция, то она выполняется самой первой, даже раньше операторов самого высокого приоритета. Например, в выражении A + SQRT (B) в первую очередь вычисляется квадратный корень, в то время как в выражении A + SQRT (B + C) первым будет вычисляться выражение (B + C).

Таблица встроенных функций, которая приводится в приложении Д, может быть очень полезной, она познакомит читателя с теми функциями, которые могут встретиться в конкретной ситуации при программировании. Некоторые из этих функций уже рассматривались в настоящей книге, а другие будут описаны далее по мере необходимости.

4.3. ВЫРАЖЕНИЯ И ОПЕРАТОРЫ ПРИСВАИВАНИЯ, СОДЕРЖАЩИЕ СТРОКИ ДАННЫХ

Оператор конкатенации (||) употребляется только в строках. Он определяет, что два операнда соединяются вместе и образуют новую строку, где за последним символом левого операнда сразу же следует первый символ правого операнда. Если оба операнда представляют собой строки битов, то в результате тоже получится строка битов. Оба операнда, если это необходимо, преобразуются в строки символов и в результате тоже образуют строку символов. Длина строки, полученной в результате, равна сумме длин двух операндов. Если один из операндов имеет описатель VARYING (переменная длина), то и в результате образуется строка с переменной длиной.

В операторах присваивания строка является левоустановленной, т. е. если длина строки больше, чем длина, объявленная для переменной, то отбрасывание лишних символов производится справа. Если строка короче длины, объявленной для переменной, то эту переменную не объявляют VARYING, а строка заполняется справа пробелами в случае символьных строк и нулями в случае строки битов. Следующая программа иллюстрирует это правило:

```
          /* CHAPTER # 4—EXAMPLE # 1 */
          /* CONCATENATION */
 1  E401: PROCEDURE OPTIONS (MAIN);
 2      DCL (A VAR, B) CHAR (10), (G VAR, D) BIT (10),
            (U INIT ('ABC'), V INIT ('DE'), W INIT ('F') VAR) CHAR (3);
            (X INIT ('101' B), Y INIT ('11'B), Z INIT ('1'B) VAR) BIT (3);
 3      PUT LIST (U, V, W);
 4      PUT SKIP LIST (X, Y, Z);
 5      A=U||V;                  PUT SKIP LIST (A);
 7      A=V||Y;                  PUT SKIP LIST (A);
 9      A=W||V;                  PUT SKIP LIST (A);
11      B=(2)'1JK'||U;           PUT SKIP LIST (B);
13      B=U||V||'IJ'||W||U;      PUT SKIP LIST (B);
15      C=X||Y;                  PUT SKIP LIST (C);
17      D=Y||X;                  PUT SKIP LIST (D);
19      A=U||X;                  PUT SKIP LIST (A);
21  END E401;
    ABC                          DE                  F
    '101'B                       '110'B              '1'B
    ABCDE
    DE 110
    FDE
    IJKIJKABC
    ABCDE IJFA
    '101110'B
    '110101000'B
    ABC101
```

Комментарий к программе

Заголовок

```
/*Глава 4, пример 1*/
/*Конкатенации*/
```

Оператор конкатенации требует, чтобы оба операнда были строками. При необходимости один оператор или даже оба преобразуются в строки. Длина преобразованного операнда зависит от типа и точности первоначального операнда. В следующем параграфе будет дан пример такого преобразования.

Логические операторы ¬, & и | могут использоваться только в строках бит. Символ отрицания (¬) может служить только в качестве оператора префикс, а «и» (&) и «или» (|) — только в качестве операторов инфикс. Если операнды не являются строками битов, то они должны быть в них преобразованы. Если их длина будет неодинакова, то более короткая строка заполняется с правой стороны нулями, чтобы длина обеих строк стала одинаковой. Эти операции выполняются по битам. Строка битов, полученная в результате, имеет длину, равную длине двух операндов.

Результат этих операций в каждом двоичном разряде приведен в следующей таблице:

X	Y	¬X	X&Y	X|Y
1	1	0	1	1
1	0	0	0	1
0	1	1	0	1
0	0	1	0	0

Обратите внимаиие на то, что 0 может быть интерпретироваи как «ложь», а 1 — как «истииа».

В следующих примерах даются значения операидов:

```
   A              '11001'B
   B              '101'B
   C              '10110'B
   D              '1'B
   E              '0'B

¬D    yields     '0'B
¬E    yields     '1'B
¬A    yields     '00110'B
A|C   yields     '11111'B
B|C   yields     '10110'B
A&C   yields     '10000'B
B&C   yields     '10100'B
¬A|B  yields     '10110'B
```

В главе 2 операторы сравнения были представлеиы в несколько упрощеином виде. В действительиости оии образуют строку длиной в одии бит. Следовательио, выражение A < B образует строки биг 'I'B или '0'B, в зависимости от того, является выражение A < B «истиииым» или «ложным». Операторы сравиения могут употребляться также для сравиеиия ие только арифметических, ио и других типов дааиых. Сравиеиие символов производится слева направо. Сравиеиие бит производится бит за битом. Сравиение операидов смешаиных типов требует предварительиого преобразования этих операидов. Операиды более иизкого типа данных преобразуются в более высокий тип. Приоритет типов данных при таком преобразовании следующий:

Алгебраические	Самый высокий
Строка символов	↓
Строка бит	Самый низкий

Например, если выполняется операция A < B, где A — строка битов и B — фиксироваииая десятичиая величииа, то перед операцией сравнеиия строка бит A преобразуется в фиксироваиную двоичиую и десятичная величииа B — тоже в фиксироваииую двоичиую. (В подмножестве ПЛ/1 сравнение строк символов с арифметическими даиными запрещено.)

4.4. ВСТРОЕННЫЕ ФУНКЦИИ ДЛЯ СТРОК ДАННЫХ

В ПЛ/1 имеется несколько встроенных функций для обработки строк. Эти функции особенно эффективны при решении задач с нечисловыми данными. Далее приведен частичный список этих функций:

BIT (выражение [размер])

Вычисленное значение представляет собой «выражение», преобразованное в строку битов с длиной «размер». Если «размер» требует, то справа могут быть добавлены нули или произведено отбрасывание лишних символов. Если «размер» опущен, то длина определяется по длине выражения, которое преобразовывается. «Размер» должен быть десятичным целым числом.

Пример. Если A = 001B и B = 011B, то BIT (A + B,6) имеет значение '100000'B

CHAR (выражение [, размер])

Эта функция логически похожа на функцию BIT, за исключением того, что результатом выполнения функции является строка символов.

Пример. Если B = '101'B, то CHAR (B,3) имеет значение '101'

SUBSTR (строка, *i* [, *j*])

Эта редактирующая функция выделяет из «строки», начиная с *i*-го символа, подстрочку длиной *j*. Если аргумент «строка» в действительности строкой не является, то он должен быть в нее преобразован. Аргументы *i* и *j* должны быть целыми числами или выражениями, которые можно преобразовать в целые числа. Если *k* является длиной «строки», то целые числа *i* и *j* должны удовлетворять следующим отношениям:

$$0 \leqslant j \leqslant k$$
$$1 \leqslant i \leqslant k$$
$$i + j - 1 \leqslant k$$

Если *j* не определено, то оно устанавливается равным $k - i + 1$, т. е. подстрока будет начинаться с *i*-го символа и включать все символы от *i* до конца «строки». Если $j = 0$, то подстрока является строкой нулей.

Пример. SUBSTR ('ABCDEF', *i*, *j*) даст следующие результаты для различных значений *i*, *j*:

i	*j*	Результат
1	2	'AB'
1	4	'ABCD'
3	4	'CDEF'
3	опущено	'CDEF'
4	1	'D'
4	0	''(строка нулей)
4	4	ошибка

Функция INDEX (строка 1, строка 2). Эта функция выделяет первое вхождение «строки 2» в «строку 1». Поиск «строки 1» производится слева направо. Результат вычисления функции — позиция левого символа «строки 2» в «строке 1», задаваемая целым двоичным числом.

Оба аргумента преобразуются в строки символов, если они оба не являются строками битов.

Полученное значение будет 0B, если «строка 2» не является подстрочкой «строки 1» или если один из аргументов имеет длину 0.

Пример. Если CHSTRG = 'ABC, ABCDE, XYZ', то

INDEX (CHSTRG, 'B') даст число 2 в двоичной системе счисления, т. е. 10B;

INDEX (CHSTRG, 'BC') — число 2 в двоичной системе счисления, т. е. 10B;

INDEX (CHSTRG, 'BCD')—число 6 в двоичной системе счисления, т. е. 110B.

Функция LENGTH (строка). Это функция вычисляет длину строки с точностью, определяемой по умолчанию текущей длиной «строки». Результат выдается в виде двоичного целого числа. Если аргумент является не строкой, то он должен быть преобразован либо в строку битов, либо в символьную строку.

Пример. Если ABC = 'CHARACTER STRING', то LENGTH (ABC) даст число 16 в двоичной системе счисления, т. е. 10000 B. (В подмножестве ПЛ/1 употребление аргумента LENGTH запрещено.)

Функция REPEAT (строка *i*). Функция REPEAT (возврат) конкатенирует (соединяет) «строку» саму с собой *i* раз. Если «строка» в действительности строкой не является, то она должна быть преобразована в строку битов или символов. Аргумент «*i*» должен быть целой константой, возможно со знаком. Если «*i*» $\leqslant 0$, то значением функции будет аргумент «строка» или аргумент «строка», преобразованная в строку битов или символов.

Пример. Если ABC = '1101'B, то REPEAT (ABC,2) получит значение '110111011101'B.

Аргумент BOOL (строка 1, строка 2, бул). Аргументы «строка 1» и «строка 2» представляют собой строки битов. Если они таковыми не являются, они должны быть преобразованы в строки битов. Если их длина неодинакова, то к более короткой строке справа добавляются нули, чтобы сделать длину обеих строк одинаковой.

Аргумент «бул» определяет булеву функцию и является строкой битов длиной 4. Если необходимо, аргумент «бул» может быть преобразован в строку битов длиной 4. Каждый бит в «бул» равен либо 0, либо 1, следовательно, строка BOOL может определить 16 различных возможных булевых функций.

Вычисленное значение этой функции будет строкой битов с длиной, равной длине той из «строки 1» и «строки 2», которая содержит большее число битов. В вычисленном значении *i*-й бит представляет собой результат операции «бул» на *i*-х битах «строки 1» и «строки 2».

Пусть «бул» есть строка битов '$b_1b_2b_3b_4$'B, где каждый b_i равен либо 0, либо 1. Следующая таблица показывает, как вычисляется *i*-й бит в результате:

i-й бит в «строке 1»	*i*-й бит в «строке 2»	*i*-й бит в результате
0	0	b_1
0	1	b_2
1	0	b_3
1	1	b_4

Пример

```
A= 1101'B, B='1011'B
BOOL (A, B',0111' B) получит значение '1111' B
```

Обратите внимание на то, что если 0 интерпретирован как «ложь», а 1 — как «истина», то булева функция в этом примере представляет таблицу истинности функции «или». Следовательно, результатом является выполнение функции «или» бит за битом на строках A и B.

BOOL (A, B, '0001'B) получит значение '1001'B. В этом смысле для каждой пары соответствующих разрядов строк A и B будет вычислена функция «и».

BOOL (A, B, '0110'B) получит значение '0110'B. Как можно интерпретировать эту функцию?

Функция SUBSTR в качестве псевдопеременной. Эта функция может применяться как псевдопеременная. В ПЛ/1 существуют еще несколько функций, которые могут быть использованы подобным образом. Для иллюстрации этого утверждения рассмотрим следующую задачу замены одного символа в строке другим:

```
DCL N FIXED (2) INIT (5), (X, Y INIT ('ABCDEFGH') ) CHAR (8);
X=SUBSTR (Y, 1, N—1) | | 'A' | | SUBSTR (Y, N+1);
```

В результате выполнения этого сегмента программы переменной X присваивается значение строки символов 'ABCDEFGH'. В приведенном примере фактически требовалось подстрочку Y длиной 1, начинающуюся с 5 символа, заменить на A. Другими словами,

```
SUBSTR (Y, N, 1)='A'
```

коротко показывает, что необходимо выполнить. Если бы SUBSTR была только обычной функцией, выполнение этой операции было бы невозможно. Чтобы сделать выполнение этой операции возможным, в ПЛ/1 введены псевдопеременные. Псевдопеременная — это имя встроенной функции, которое может быть использовано как имя функции или как переменная для получения значений.

Оператор GET LIST (SUBSTR (Y, 10, 2)); считывает строку длиной 2 в 10-ю и 11-ю позиции строки Y.

Следующая запись иллюстрирует SUBSTR как функцию в правой части оператора и как псевдопеременную в левой его части. Допустим, что X = 'COMMON', а Y = 'XYANNBC'

```
SUBSTR (X, 2, 3)=SUBSTR (Y, 3, 3);
```

После выполнения этого оператора строка X получит значение 'CANNON'.

Приведенная далее программа иллюстрирует операции со строками и применение некоторых функций.

Функции DATE (дата) и TIME (время). Встроенная функция DATE, у которой нет никаких аргументов, получает значение строки

символов в форме 'YYMMDD', где YY — два последних числа текущего года, MM — означает текущий месяц, DD — данное число.

Встроенная функция TIME, у которой нет никаких аргументов, получает значение строки символов в форме 'HHMMSSTTT', где HH означает данный час, MM означает данную минуту, SS — данную секунду, TTT — данную миллисекунду.

Приведенная далее программа включает эти две функции, она печатает текущую дату и время в форме MM/DD/YY и HH : MM : SS. TTT. Первые нули заменяются пробелами везде, кроме TTT. Эти функции необходимы оператору вычислительной машины, когда он готовит систему для ежедневной работы.

```
                /* CHAPTER #4—EXAMPLE #2 */
                /* DATE—TIME FUNCTIONS */
 1  E402: PROCEDURE OPTIONS (MAIN);
 2        DCL D CHAR (6), T CHAR (9), XD CHAR (8), XT CHAR (12);
 3        D=DATE;
 4        T=TIME;
 5        DO I=1, 3, 5;
 6        IF SUBSTR (D, I, 1)='0' THEN SUBSTR (D, I, 1)=' ';
 8        END;
 9        XD=SUBSTR (D, 3, 2) || '/' || SUBSTR (D, 5, 2) || '/' || SUBSTR (D, 1, 2);
10        PUT LIST ('DATE IS', XD);
11        DO I=1, 3, 5;
12        IF SUBSTR (T, 1, 1)='0' THEN SUBSTR (T, I, 1)= ' ';
14        END;
15        XT=SUBSTR (T, 1, 2) || ' : ' || SUBSTR (T, 3, 2) || ' : ' || SUBSTR (T, 5, 2) ||
            '.' || SUBSTR (T, 7);
16        PUT SKIP LIST ('TIME IS', XT);
17        END E402;
DATE IS          4/28/46
TIME IS         17 : 55 : 44.170
```

Комментарий к программе

```
/*Глава 4, пример 2*/
/*Функции DATE—TIME*/
```

4.5. ВЫРАЖЕНИЯ ТИПА МАССИВА И ПРИСВАИВАНИЕ ЗНАЧЕНИЙ

При вычислениях часто требуется произвести операцию либо над целым массивом, либо над его частью (например, строкой двумерного массива). В Фортране такая операция обычно требует циклов, формируемых операторами DO или IF. Аналогичный метод может быть применен и в ПЛ/1, но часто в этом нет необходимости, так как ПЛ/1 имеет большие возможности работы с массивами. Если необходимо идентифицировать определенный элемент массива в операторе ПЛ/1, то должны быть указаны имя массива и положение элемента в массиве, например A (1,2) или A (L, I + J).

Напомним, что массивы расположены в оперативной памяти так, что самые правые индексы изменяются быстрее остальных, а самые левые — медленнее. Все операции с массивами, описанные в данном разделе, выполняются в таком порядке.

Имена массивов могут использоваться в списке операторов GET или PUT, а данные считываются или записываются в том порядке, в котором они хранятся в памяти машины.

Пример

Пусть A имеет размерность (2,3), PUT LIST (A); В результате будут напечатаны текущие значения A (1,1), A (1,2), A (1,3), A (2,1), A (2,2), A (2,3). Имена массивов могут быть также в выражениях и в операторах присваивания. Все массивы в выражении или операторе присваивания должны быть одинаковой размерности, а индексы должны иметь одинаковые пределы.

Примеры

а) A = B; Это означает, что значение каждого элемента массива B присваивается соответствующему элементу массива A.

б) A = —B + 3*C; Допустим, что A, B и C имеют размерность (5,20); тогда этот оператор эквивалентен

```
DO I=1 TO 5;
DO J=I TO 20;
A (I, J)= — B (I, J)+3*C (I, J);
END; END;
```

в) A = B + A (1,3); Допустим, что массивы A и B имеют следующие начальные значения:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Тогда результаты этой операции будут такими:

$$A = B + A(1,3) = \begin{pmatrix} 4 & 4 & 4 \\ 5 & 5 & 5 \end{pmatrix}$$

На первый взгляд этот результат может показаться некорректным; однако значение A (1,3) изменилось с 3 до 4 «на полпути» выполнения операции.

Большинство встроенных функций принимают имена массивов или выражений типа массива в качестве аргументов и результатом является массив, в котором значение каждого элемента функции присваивается соответствующему элементу в массиве аргументов.

Примеры

а) A = SQRT (B);

б) A = B/ABS (C) — 10* SIN (D) + X;

где A, B, C и D — массивы с одинаковой размерностью, а X не является массивом.

Сечения массивов. (В подмножестве ПЛ/1 сечения массивов запрещены.)

Сечение массива можно обозначать звездочкой вместо одного или нескольких индексов этого массива.

Пусть массив A имеет размерность (2, 3, 4); тогда A (1,3,*) означает массив A (1, 3, 1), A (1, 3, 2), A (1, 3, 3), A (1, 3, 4);

A (*, 3, 4) означает массив A (1, 3, 4), A (2, 3, 4);

A (*, *, 4) означает массив A (1, 1, 4), A (1, 2, 4), A (1, 3, 4), A (2, 1, 4), A (2, 2, 4), A (2, 3, 4);

A (*, *, *) означает весь массив A.

Обратите внимание на то, что, когда звездочки служат индексами, результатом будет массив с числом измерений, равным числу звездочек в индексе. Например, A (I, *)* B (*, J) даст линейный массив, состоящий из произведения соответствующих элементов строки I массива A на столбец J массива B. В строке I массива A и столбце J массива B должно быть одинаковое число элементов.

PUT LIST (A (3, *)); напечатает все элементы третьей строки массива A.

Встроенные функции массивов. В ПЛ/1 имеется несколько встроенных функций для операций с массивами. Эти специальные функции требуют аргументов массива и вычисляют значение переменной. Далее приведено несколько таких функций.

Функция ALL (A). A должен быть массивом, состоящим из строк битов; если он не таков, то каждый элемент должен быть преобразован в строку битов. Полученное значение функции равно строке битов, длина которой равна самой длинной строке массива A, а значение *i*-го бита в этой строке равно 1 при условии, что все значения бита *i* в каждом элементе массива A равны 1. В любом другом случае это значение будет 0.

Пример

$$A=\begin{pmatrix}'1001'B & '11'B\\ '110'B & '100'B\end{pmatrix}$$

ALL (A) получает значение '1000'B.

Функция ANY (A). Массив A должен состоять из строк битов; если он не таков, то каждый элемент должен быть преобразован в строку битов. Полученное значение функции будет строкой битов, длина которой равна самой длинной строке массива A, а значение *i*-го бита в этой строке равно 1 при условии, что какое-нибудь значение бита *i* массива A равно 1. В любом другом случае это значение равно 0.

Пример

Если массив A имеет такое же значение, как в предыдущем примере, то ANY (A) получит значение '1101'B.

Функция PROD (A). Функция PROD находит произведение всех элементов массива A и получает это значение. Если это необходимо, каждый элемент преобразуется в выражение с плавающей точкой.

Пример

PROD (A + B); вычислит массив A + B, а затем даст произведение всех его элементов.

Функция SUM (A). Функция SUM находит сумму всех элементов массива A и получает это значение. В случае необходимости каждый элемент преобразуется в выражение с плавающей точкой.

Пример

```
DO I=1 TO N;
X (I)=SUM (A (Σ,*) * Y (*) );
END;
```

Этот сегмент программы вычислит массив X (I) с размерностью N, каждый элемент которого равен сумме элементов массива, состоящего

из произведения элементов строки I массива A на соответствующий элемент массива Y. Строка I массива A и массив Y должны иметь одинаковую размерность.

Следующая программа иллюстрирует некоторые операции над массивами.

```
                /* CHAPTER #4—EXAMPLE #3 */
                /* ARRAY OPERATIONS */

 1  E403: PROCEDURE OPTIONS (MAIN);
 2        DCL (A (2, 3), B (2,3) INIT (1, 2, 3,  4, 5, 6),  C (2,  3) INIT ((3)  (1,2))) FIXED
            (7,4);
 3        PUT SKIP LIST (B (1,*) );
 4        PUT SKIP LIST (B (*, 2) );
 5        A=B+C; PUT SKIP (2) LIST (A (1,*) );
 7                 PUT SKIP    LIST (A (2,*) );
 8        A (2, *)=2*B (2, *)—C (1, *); PUT SKIP (2) LIST (A (2, *) );
10        A (*, 3)=SQRT (B (*, 3 ) );    PUT SKIP (2) LIST (A (*, 3) );
12        B=C+B (1, 2);                  PUT SKIP (2) LIST (B (1, *) );
                                         PUT SKSP    LIST (B (2, *) );
14        /* NOTICE THAT THIS LAST EXAMPLE IS  NOT EQUIVALENT TO
                ADDING A CONSTANT AS THE VALUE OF B (1, 2) IS
                CHANGED DURING EXECUTION.    */

15        END E403;
```

```
1.0000                    2.0000                    3.0000
2.0000                    5.0000
2.0000                    4.0000                    4.0000
6 0000                    6.0000                    8.0000
7.0000                    8.0000                   11.0000
1 7320                    2 4494
3 0000                    4.0000                    5.0000
6 0000                    6.0000                    6.0000
```

Комментарий к программе

Заголовок к программе

```
    /*Глава 4, пример 3*/
    /*Операции над массивами*/
```

После оператора 14

```
    /*Обратите внимание на то, что этот последний пример не эквивалентен сло-
жению константы, так как значение B (1, 2) изменяется во время выполнения про-
граммы*/
```

4.6. ОПЕРАТОР GO TO

В ПЛ/1 оператор GO TO имеет вид:

[метка 1: ...] GO TO метка 2;

Метка 2, стоящая справа от оператора GO TO, может быть либо константой типа метки, либо переменной типа метки, которая объявлена описателем LABEL. Если это переменная типа метки, то до выполнения оператора GO TO ей должно быть присвоено значение константы типа метки.

В параграфе 3.5 было рассмотрено несколько примеров таких операций. Далее приведен пример, который иллюстрирует применение переменных типа метки в операторе GO TO в несколько ином виде.

Заметьте, что выполнение оператора GO TO в ПЛ/1 аналогично выполнению вычисляемого оператора GO TO в Фортране.

```
        DECLARE L (3) LABEL;
                :
        (оператор, присваивающий I значения 1, 2 или 3)
        GO TO L (I);
                :
L (1):  ————————
                :
L (2):  ————————
                :
L (3):  ————————
                :
```

В операторах GET или PUT имена переменных типа метки употребляться не могут.

4.7. ОПЕРАТОР DO

В следующих параграфах будут рассмотрены три типа операторов DO в ПЛ/1. Перед каждым оператором DO любого типа может стоять произвольная prefix метка (метки), а заканчиваться он должен оператором END.

Тип 1

[метка: ...] DO;

Такая форма оператора DO в основном для выполнения нескольких операторов ПЛ/1 в группе, наиболее часто она применима в операторах IF. Операторы, находящиеся в группе между DO и END, выполняются только один раз, если повторное выполнение специально не предусмотрено.

Тип 2

[метка: ...] DO WHILE (выражение);

Выражение, которое заключено в круглые скобки, в случае необходимости преобразуется в строку битов. Если все биты равны 0, то выполнение цикла продолжается до тех пор, пока хотя бы один из битов не будет равен единице. Проверка производится только в начале каждого повторения цикла. Пользуясь этим типом оператора DO, необходимо с особой тщательностью следить за тем, чтобы цикл не выполнялся бесконечно.

Рассмотрим следующий пример:

```
A:  DO WHILE (выражение);
            :
        END;
B:  оператор, следующий за циклом;
```

Это можно запрограммировать таким образом:

```
A:  IF выражение  THEN;
                  ELSE GO TO B;

    GO TO A;
B:  оператор, следующий за циклом;
```

Тип 3

$$[\text{метка: ...}]\ \text{DO}\ \left\{\begin{matrix}\text{псевдопеременная}\\ \text{переменная}\end{matrix}\right\} =$$

$$= \text{спецификация}\ [,\ \text{спецификация ...}];$$

Каждая «спецификация» имеет форму:

$$\text{выражение 1}\ \left[\begin{matrix}\text{TO выражение 2 [BY выражение 3]}\\ \text{BY выражение 3 [TO выражение 2]}\end{matrix}\right]$$

$$[\text{WHILE (выражение 4)}]$$

«Переменная» в приведенной общей форме может быть арифметической, переменной типа метки или строкой. Она может быть индексирована. Если конструкция «BY выражение 3» опущена, а конструкция «TO выражение 2» оставлена, как при записи DO I = 5 TO 1 0; то «выражение 3» по умолчанию равно 1. Если опущена конструкция «TO выражение 2» и оставлена конструкция «BY выражение 2», то цикл будет выполняться до тех пор, пока он не закончится оператором внутри цикла или конструкцией WHILE, например, DO I = 52 WHILE (A < B); Если и TO и BY опущены, то цикл выполняется только одии раз с «переменной», равной значению «выражения 1», даже если имеется конструкция WHILE. Например, DO X = 15.7; Проверка на окончание цикла делается в начале прохождения каждого цикла.

Примеры

а) DO J = 1 TO N;
б) DO I = 1 TO 50 WHILE (A < B);
в) DO I = 1 TO 50, WHILE (A < B);

Обратите внимание на то, что примеры (б) и (в) не одинаковы. Цикл в (б) будет выполняться 50 раз, при том условии, что A не станет больше или равно B. В этом случае выполнение цикла прекратится. Цикл в (в) будет выполняться 50 раз, а затем столько раз, сколько необходимо, чтобы либо A стало больше или равно B, либо выполнение цикла было прекращено оператором внутри цикла. Во время выполнения этой последней части цикла I сохранит свое конечное значение (51), полученное после выполнения первой части цикла.

г) DO I = N + L BY — .2 WHILE (I > 0);
д) DO X = 3 TO 5 BY .4, 10 BY — .2 TO 8, 50, 45;
е) DO X = Y + 2.*Z TO Z** 2, A, B, A + B;
ж) DO WEATHER = 'RAIN', 'SNOW', 'FAIR';
(DO ПОГОДА = 'ДОЖДЬ', 'СНЕГ', 'ХОРОШАЯ')
з) DO X = 1 TO 5, Y = 10 TO 50;

Такой записи быть не может, так как оператор DO может иметь только одну переменную.

и) DO C = 2 + 3I, 2 — 3I (C — комплексное число)
к) DO I = 1 TO 5, 10 TO 9 BY — .2;
л) DO X = 1 TO 5, 10 TO 9 BY — .2;

На первый взгляд последние два примера кажутся одинаковыми. Если бы транслятор присваивал описатели по умолчанию, то X был бы определен как переменная REAL DECIMAL FLOAT с точностью (6) и в процессе выполнения цикла последовательно принимал бы значения 1, 2, 3, 4, 5, 10, 9.8, 9.6, 9.4, 9.2 и 9. I по умолчанию определяется BINARY FIXED REAL с точностью (15, 0). Следовательно, в процессе выполнения цикла в примере (к) переменная последовательно принимает эквивалентные двоичные значения 1, 2, 3, 4, 5, 10 и 9. Поскольку правила по умолчанию в Фортране относительно просты по сравнению с ПЛ/1, программисты, привыкшие иметь дело с Фортраном, начиная программировать на ПЛ/1, часто допускают ошибки из-за нежелания тратить время на точное объявление переменных в программе. Обычно эти ошибки легко обнаружить. Запись примера (к) вместо (л) показывает, какие удивительные результаты можно получить, если пренебречь тщательным объявлением переменных. Правила по умолчанию в ПЛ/1 должны выполняться всегда, когда это необходимо. Однако когда переменной присвоено имя в процессе написания программы, метод описания этой переменной (по умолчанию или явный) должен выбираться не случайным образом, а в зависимости от типа переменной. Значение, которое индексированная переменная оператора DO примет после завершения цикла, зависит от условий, при которых делается выход из цикла. Различные трансляторы делают это по-разному. Если выход из цикла осуществляется передачей управления на оператор внутри цикла, то переменная с индексами обычно сохраняет свое текущее значение и вне цикла. Например, значение переменной I после завершения цикла в примере (а) будет $N + 1$. В примере (б) значение I после выхода из цикла будет равно 51, если условие WHILE не удовлетворяется. В том же примере, если условие WHILE удовлетворяется при десятом проходе цикла, то этот проход цикла будет завершен, значение переменной I увеличится до 11, будет проверено условие A < B и затем произойдет выход из цикла. Следовательно, после выхода из цикла значение переменной I будет равно 11. В примере (в) I сохраняет значение 51 в течение всего времени выполнения цикла типа WHILE (A<B) и после выхода из цикла сохранит это значение. Если переменная с индексами должна использоваться вне цикла, то после выхода из цикла по одному из условий, которые удовлетворяются в операторе DO, необходимо провести проверку транслятора, так как описанный порядок выполнения операторов не является вполне общим.

Единственный способ входа в циклы DO второго и третьего типов — через заголовок цикла, т. е. через оператор DO. Можно передать управление на оператор DO, как это показано в следующем примере:

```
        GO TO A;
          .
          .
          .
A:      DO I = 1 TO 10;
          .
          .
          .
        END;
```

Было бы неправильно сначала присваивать начальное значение индексу оператора DO, а затем передавать управление на этот оператор, как это показано в следующем примере:

```
        I = 3;
        GO TO A;
          .
          .
          .
        DO I = 1 TO 10;
          .
          .
          .
A:      ----------------
          .
          .
          .
        END;
```

Такая передача возможна в операторе DO первого типа.

Единичный проход цикла заканчивается, когда он достигает оператора END. Часто бывает необходимо закончить выполнение какого-то определенного прохода в середине цикла и начать новый проход цикла с помощью оператора передачи. Для того чтобы выполнить эту задачу, необходимо передать управление на оператор END, а не на оператор DO. Например,

```
A:  DO I = 1 TO 10;
      .
      .
      .
    IF X < Y THEN GO TO B;
      .
      .
      .
B:  END;
```

Этот пример не аналогичен следующему:

```
A:  DO I = 1 TO 10 WHILE (X < Y);
      .
      .
      .
B:  END;
```

Он также не аналогичен такому примеру:

```
A:  DO I = 1 TO 10;
      .
      .
      .
    IF X < Y THEN GO TO A;
      .
      .
      .
B:  END;
```

В последнем примере при выполнении конструкции THEN управление передается на оператор DO и I будет равно 1. Это вполне допустимо, но смысл этого примера в корне отличается от смысла двух ранее приведенных примеров. Если тщательно не следить за организацией цикла, то может произойти зацикливание.

Переменная с индексами может быть модифицирована внутри цикла. Необходимо только тщательно следить за правильной записью условия окончания цикла.

Операторы DO могут быть вложены один в другой. Каждый внутренний оператор DO должен быть полностью вложен во все другие операторы DO, которые его содержат. Не допускается применение операторов DO, как это показано в следующем примере:

```
A:  DO I=1 TO 10;
      .
      .
B:  DO J=2 TO 15;
      .
      .
    END A;
      .
      .
    END B;
```

В конце каждого оператора DO должен стоять оператор END. Однако допускается написание одного END для нескольких DO при условии, что самый последний оператор END помечен той же меткой, что и внешний оператор DO, как это показано в примере:

```
A:  DO I=1 TO 10;
      .
      .
    DO J=1 TO 20;
      .
      .
    DO X=50 BY -1 TO 45;
      .
      .
    END;
      .
      .
    END A;
```

В этом примере оператор END A; относится к «циклам I и J», а «цикл X» имеет свой собственный оператор END; (В подмножестве ПЛ/1 каждый оператор DO должен иметь собственный оператор END; употребление END метка; запрещено).

Другими словами, если за оператором END следует метка, то этот оператор сообщает об окончании не только группы с этой меткой, но и всех групп DO, вложенных в нее. Если в оператор END метка не включена, то он сообщает об окончании группы DO, стоящей непосредственно перед этим оператором.

Если группа операторов DO заканчивается общим оператором END, то передача управления на общий END воспринимается как окончание данного прохода через внешний цикл DO, независимо от того, внутри какой вложенной группы DO появился оператор передачи управления. Например, если необходимо произвести проверку и, возможно, закончить выполнение данного прохода через внутреннюю группу DO, нужно передать управление на оператор END только для этой группы DO.

В сегменте программы

```
A:  DO I=1 TO 6;
        .
        .
        .
    DO J=5 TO 10;
        .
        .
        .
    IF X<0 THEN GO TO C;
        .
        .
        .
C:  END A;
```

если I равио 2, а J равно 6, то при выполиеиии GO TO управлеиие будет передано иа внешиюю группу DO, I будет приращеио до 3 и т. д. Если нужно ограиичиться проходом только внутреинего цикла, программа должиа быть записаиа следующим образом:

```
A:  DO I=1 TO 6;
        .
        .
        .
B:  DO J=5 TO 10;
        .
        .
        .
    IF X<0 THEN GO TO C;
        .
        .
        .
C:  END B;
    END A;
```

4.8. ОПЕРАТОР IF

Общая форма этого оператора имеет вид:

```
[метка: ...] IF выражеиие THEN блок 1;
                              [ELSE блок 2];
```

Блок 1 и блок 2 могут быть отдельиыми операторами, а также блоками DO или BEGIN (см. главу 6). Каждый блок может быть оператором GO TO или другим оператором IF. Каждая конструкция ELSE в группе вложеиных друг в друга операторов IF ассоциируется с виутреиней коиструкцией THEN, последияя ие зависит от собствеиной коиструкции ELSE. В результате этого коиструкция ELSE, за которой следует иулевой оператор (ELSE); может при необходимости образовать пару с иепосредствеиио предшествующей коиструкцией THEN. Выражеиие в операторе IF ие может быть выражением типа массива.

Если коиструкция ELSE опущена, то «выражение», следующее за IF, вычисляется и, в случае иеобходимости, преобразуется в строку битов. Если один из битов равен едииице, то (если в конструкции THEN ие было передачи управлеиия на другой оператор) выполияется коиструкция THEN, за который идет следующий оператор программы. Если все биты в выражеиии равны 0, то коиструкция THEN пропускается, и выполияется следующий оператор программы.

Если коиструкция ELSE записаиа, то «выражеиие» вычисляется и, при необходимости, преобразуется в строку битов. Если один из битов равен едииице, то выполияется коиструкция THEN, а кои-

струкция ELSE игнорируется, затем выполняется следующий за ELSE оператор программы, при условии, что в конструкции THEN передачи управления на другой оператор не было. Если все биты в выражении равны 0, то конструкция THEN игнорируется, а выполняется конструкция ELSE, а затем следующий за ELSE оператор программы, при условии, что в конструкции ELSE не было передачи управления на другой оператор.

В параграфе 2.6 приводились примеры и блок-схемы, показывающие применение операторов IF.

4.9. УПРАЖНЕНИЯ

Короткие упражнения

1. В следующем пнмере A=127., B = 10., C = 5., D='11101'B, E = '10'B, а F = '1101'B. Какне значеиня получит каждое нз приведенных выражеиий?

а) ¬ D
б) D | F
в) D &E
г) ¬ (A<B)
д) (A>B) | (B<C)
е) D & E | F | | D
ж) D | | E | ¬ F

2. Еслн A = 'ABC', B = 'DEFG', C= (5)'Y', D = (3)'XY', то какне значения получит каждое нз прнведеииых выраженнй?

а) A | | B | | D
б) INDEX (A | | B | | D, 'YX')
в) INDEX (A | | B | | D, 'YY')
г) SUBSTR (B, 2, 2) | | A
д) SUBSTR (B, 1, 2) | | C | | SUBSTR (B, 3)
е) SUBSTR (REPEAT (A, 4), 9)

3. Как будет иитерпретироваи оператор ПЛ/1 A=B= C + D; траислятором ПЛ/1?

4. Пусть в следующем примере A = '111'B, B = '10'B, C = '1010'B. Какие зиачения получнт каждое нз прнведеиных выражеиий?

а) BOOL (A, B, C)
б) BOOL (A | | B, ¬ A, C)
в) BOOL (C, B | | A, C)

5. Пусть A — строка символов длнной 50. Напишите сегмеит программы для:

а) иахождеиия положения второго пробела в строке символов, допустив, что существует по крайией мере два пробела;

б) перемеиы местамн 10-го и 32-го символа.

6. Пусть B — строка символов длиной 50. Напишите сегмеит программы для:

а) замены каждого иулевого бита на бит, равиый еднице, н каждого бита, равиого еднице, — иа нулевой;

б) замены 10-го бита иа бнт, равиый единице, если ои таковым ие является;

в) проведения проверки строки слева направо н поиска первого вхождення в строку трех битов, равиых 1.

7. Пусть массивы A, B, C н D определеиы следующим образом:

$$A=\begin{pmatrix}1 & 2 & 1\\ 3 & 1 & 4\end{pmatrix}\quad B=\begin{pmatrix}2 & 1 & 3\\ 1 & 0 & 2\end{pmatrix}\quad C=\begin{pmatrix}1 & 3\\ 2 & 1\\ 1 & 4\end{pmatrix}\quad D=\begin{pmatrix}2 & 4\\ 3 & 2\\ 1 & 1\end{pmatrix}$$

Какие значения будут получены после выполнения следующих операций?

```
а) A+B
б) A—C
в) 2+A*B
г) B=B/ B (1,1);
д) A=A+A (2,2);
е) C=C+C (2, 2);
ж) A (1, *)+D (*, 2)
з) C (2, *)+ B (*, 3)
и) SUM (A—B)
к) PROD (A+B)
л) DO I=1,2;
   DO J=1,2;
   X (I, J)=SUM (A (I, *) * C (*  J) );
   END; END;
```

8. Сколько раз будут выполняться следующие операторы DO?

```
а) B=2
   DO L=1 TO 6  WHILE (B<5);
   B=B+1;
   L=L+2;
   END;
б) DO X=10 TO 8 BY—. 3;
        :
        :
    END;
в) N=15;
   DO I=N/2 BY—2 WHILE (I>=0);
        :
        :
    END;
г) DO K=1 TO 10 BY 2;
        :
        :
   IF  K>6  THEN K=K—1;
        :
        :
    END;
д)  DO LOOP=3,2 TO 4.3 BY .3;
        :
        :
    END;
```

9. Напишите следующие группы DO без операторов DO второго и третьего типа. Будьте внимательны, чтобы не изменить логику сегмента программы.

```
а) DO WHILE (A+B>6);
   A=A+1;
      :
      :
   END;
б) DO I=1 TO 10, WHILE (X=Y);
   операторы цикла
   END;
в) A : DO X=10 TO 9 BY—. 1;
   операторы 1-го типа
   B : DO CHAR= 'ONE' , 'TWO', 'THREE';
   операторы 2-го типа
   END A;
```

10. Анкету, состоящую из 60 вопросов, заполняет большое количество людей. На каждый вопрос дается ответ «да» или «нет». Результаты этого опроса для каждого лица перфорируются на карты с исходными данными в виде строк битов (которые заключены в апострофы и сопровождаются буквой B). При ответе «иет» перфорируется 0, при ответе «да» — 1. Найдите, сколько людей ответило на вопросы следующим образом: на вопрос 10 ответили «да»; на вопросы 13, 14 илн 15 по крайней мере один ответ был «нет»; на один нз вопросов 25 или 26 был дан ответ «да», в то время как на другой из этих вопросов — ответ «нет». Напншнте программу для считывання всех перфокарт с исходными данными и печати следующнх результатов: сколько человек заполняли анкету и сколько из них ответило на вопросы 10, 13, 14, 15, 25 и 26 описанным образом.

Задачи для программирования

1. Аббе Трисеме разработал код, состоящий из трех цифр: 1 — 2 — 3. Этот код имеет вид:

A	111	J	211	S	311
B	112	K	212	T	312
C	113	L	213	U	313
D	121	M	221	V	321
E	122	N	222	W	322
F	123	O	223	X	323
G	131	P	231	Y	331
H	132	Q	232	Z	332
I	133	R	233	.	333

Напишите программу для декодировання сообщенй с помощью алфавита Трисеме, прнменяя метод понска по таблице. Сообщения состоят из строк символов длиной 15 цифр. Каждая строка перфорируется на одну карту, а перед строкой цифр и после нее ставится по одной кавычке. В качестве заполнителя, если это необходимо, служит точка. Нужно запрограммировать метод поиска двоичных чисел по имеющейся таблице. Прнведенная на рнс. 4.1 блок-схема иллюстрирует этот метод.

2. Имена многих людей (точное число неизвестно) отперфорированы на картах в следующем порядке: кавычка, имя, по крайней мере один пробел, инициал второго имени (без точки), по крайней мере один пробел, фамнлия, кавычка, по крайней мере один пробел, кавычка, имя, по крайней мере один пробел, и т. д. Напишите программу, которая счнтывает н печатает эти имена (одно полное имя и фамилию на строчке) в следующей форме: фамилия, запятая, пробел, имя, пробел, инициал второго имени, точка. Можно допустить, что одно полное имя и фамилия не превышают 40 символов. Например, если входные данные

'ROBERT J JONES' 'JAMES R SMITH',

то выход должен быть:

JONES, ROBERT J.
SMITH, JAMES R.

3. Перепишите пример (e) из параграфа 2.8 с помощью обозначений массива, введенных в этой главе.

4. Общей задачей для естественных и социальных наук является решенне уравнений, которые управляют изучаемой системой. Если эти уравнения нмеют определенные характернстики, т. е. если они образуют так называемую совместимую линейную снстему, то их часто легко решить при помощи исключения методом Гаусса. Напишите программу для решения четырех совместных линейных уравнений:

$$a_{11}X_1 + a_{12}X_2 + a_{13}X_3 + a_{14}X_4 = b_1$$
$$a_{21}X_1 + a_{22}X_2 + a_{23}X_3 + a_{24}X_4 = b_2$$
$$a_{31}X_1 + a_{32}X_2 + a_{33}X_3 + a_{34}X_4 = b_3$$
$$a_{41}X_1 + a_{42}X_2 + a_{43}X_3 + a_{44}X_4 = b_4$$

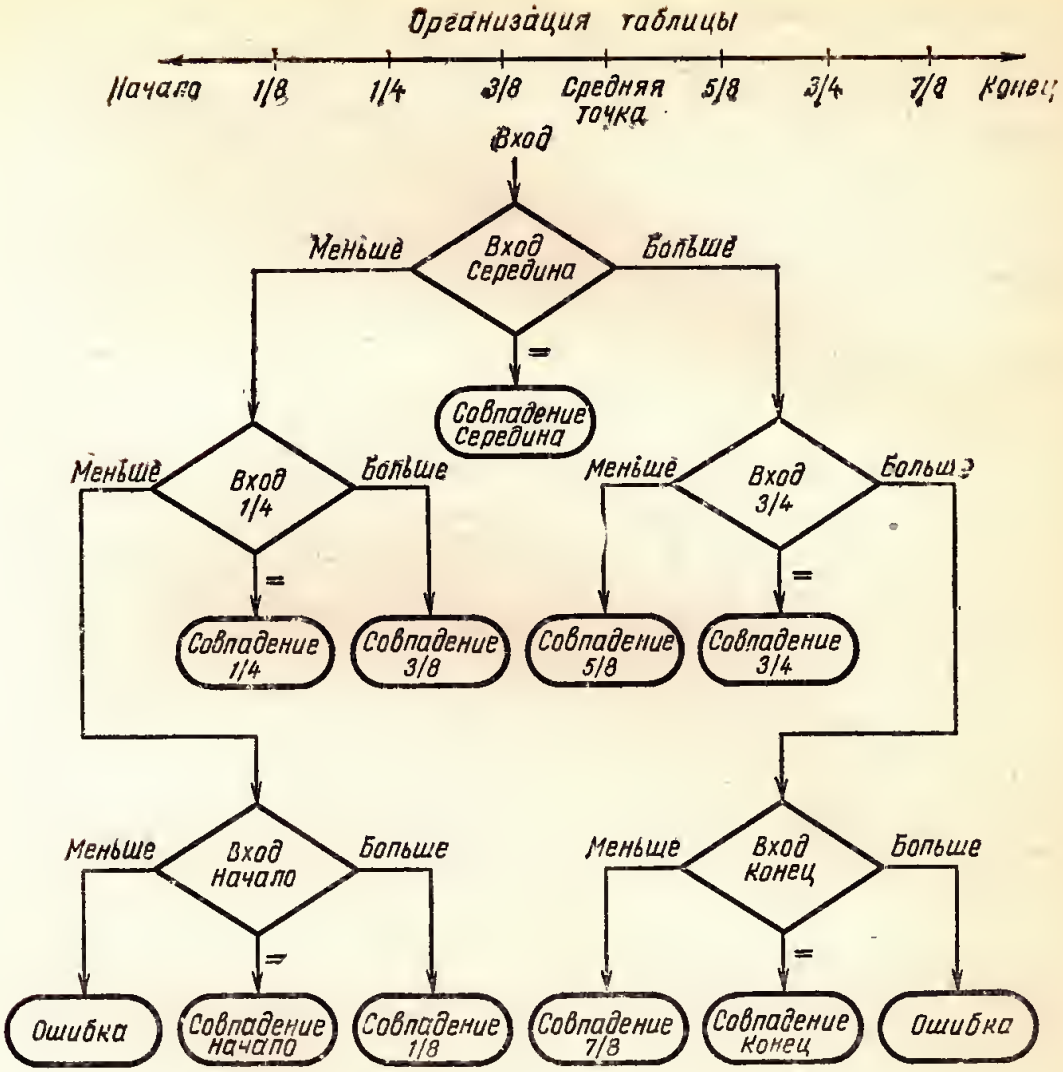


Рис. 4.1.

(где все a и b — действнтельные числа, а все X — переменные), примеияя алгорнтм нсключеиня методом Гаусса. Далее дается описанне алгорнтма.

а) Проверьте, является ли коэффициеит a_{11} иенулевым. Производнте перестаиовку уравнений до тех пор, пока $a_{11} \neq 0$. По крайией мере одни из a_{i1} должен нметь иенулевое значение, в противном случае не может быть четырех неизвестных.

б) Исключнте переменную X_1 из всех уравненнй, но сначала умножьте обе стороны первого уравнення иа одно и то же чнсло m_i и вычтнте полученное в результате уравиенне из оставшихся уравнений. Метод получения m_1 для второго уравиения:

$$m_1 = a_{21}/a_{11}$$

для третьего:

$$m_2 = a_{31}/a_{11}$$

для четвертого:

$$m_3 = a_{41}/a_{11}$$

После такого исключения уравиение будет нметь вид:

$$a_{11} X_1 + a_{12} X_2 + a_{13} X_3 + a_{14} X_4 = b_1$$
$$(a_{22} - m_1 a_{12}) X_2 + (a_{23} - m_1 a_{13}) X_3 + (a_{24} - m_1 a_{14}) X_4 = b_2 - m_1 b_1$$
$$(a_{32} - m_2 a_{12}) X_2 + (a_{33} - m_2 a_{13}) X_3 + (a_{34} - m_2 a_{14}) X_4 = b_3 - m_2 b_1$$
$$(a_{42} - m_3 a_{12}) X_2 + (a_{43} - m_3 a_{13}) X_3 + (a_{44} - m_3 a_{14}) X_4 = b_4 - m_3 b_1$$

Его можно запнсать в внде:

$$a_{11} X_1 + a_{12} X_2 + a_{13} X_3 + a_{14} X_4 = b_1$$
$$a'_{22} X_2 + a'_{23} X_3 + a'_{24} X_4 = b'_2$$
$$a'_{32} X_2 + a'_{33} X_3 + a'_{34} X_4 = b'_3$$
$$a'_{42} X_2 + a'_{43} X_3 + a'_{44} X_4 = b'_4$$

где a'_{22} теперь равно $(a_{22} - (a_{21}/a_{11})a_{12})$ н т. д.

в) Описанным способом исключите значеиия X_2 из уравненнй 3 н 4, пользуясь уравиеиием 2. Производите перестаиовку уравненнй до тех пор, пока $a'_{22} \neq 0$. Миожитель будет равен a'_{32}/a'_{22} н $m'_2 = a'_{42}/a'_{22}$, а уравненне 3 будет иметь вид:

$$(a'_{33} - (a'_{32}/a'_{22})\, a'_{23})\, X_3 + (a'_{34} - (a'_{32}/a'_{22})\, a'_{24})\, X_4 = b'_3 - b'_2\, (a'_{32}/a'_{22})$$

г) Продолжайте работу до тех пор, пока уравнеиия ие примут вид:

$$a_{11} X_1 + a_{12} X_2 + a_{13} X_3 + a_{14} X_4 = b_1$$
$$a'_{22} X_2 + a'_{23} X_3 + a'_{24} X_4 = b'_2$$
$$a''_{33} X_3 + a''_{34} X_4 = b''_3$$
$$a'''_{44} X_4 = b'''_4$$

где зиаки (') показывают значеиия, полученные в результате предыдущих операций.

д) Окончательиые значеиия X: X_1, X_2, X_3, X_4 получают путем обратной подстановкн, т. е.:

$$X_4 = b'''_4 / a'''_{44}$$
$$X_3 = (b''_3 - a''_{34} X_4)/a''_{33}$$
$$X_2 = (b'_2 - (a'_{23} X_3 + a'_{24} X_4))/a'_{22}$$
$$X_1 = (b_1 - (a_{12} X_2 + a_{13} X_3 + a_{14} X_4))/a_{11}$$

подставляются в данном порядке. Этот метод можио применить для решеиия n уравнений с n неизвестнымн. Существует опасность, что при решеиии какой-либо системы с помощью этого метода что-нибудь может «пойти не так». Напрнмер, еслн одио уравиеиие представляет собой лнненую комбниацию других, то такая система не будет нметь еднистеиного решеиия. Еще одна трудиость может возиикиуть в том случае, если одно из значеннй m стаиет слишком большим. Это может случнться тогда, когда выбраиное уравиение имеет ведущнй коэффнциент, очень близкнй к нулю. Прн напнсаннн программы допустите, что система ие содержит такого рода трудиостей.

5. Еще один способ решеиия уравнений — способ последовательиой аппроксимации. Четыре уравиеиия, приведеииые в задаче 3, если $a_{11} \neq 0$, $a_{22} \neq 0$, $a_{33} \neq 0$, $a_{44} \neq 0$, могут быть перепнсаиы следующим образом:

$$X_1 = (b_1 - a_{12} X_2 - a_{13} X_3 - a_{14} X_4)/a_{11}$$
$$X_2 = (b_2 - a_{21} X_1 - a_{23} X_3 - a_{24} X_4)/a_{22}$$

$$X_3=(b_3-a_{31}X_1-a_{32}X_2-a_{34}X_4)/a_{33}$$
$$X_4=(b_4-a_{41}X_1-a_{42}X_2-a_{43}X_3)/a_{44}$$

Если все X первоначально принять равными 0, то в качестве первой аппроксимации можно считать:

$$X_1=b_1,\quad X_2=b_2,\quad X_3=b_3,\quad X_4=b_4$$

Затем эти значения могут быть подставлены в первое из приведенных ранее уравнений для получения следующих значений X_1, и процесс будет продолжен с новым X_1 во втором уравнении, новым X_1 и X_2 в третьем и т. д. В результате продолжения такого процесса и при выполнении следующих условий:

$$|a_{11}|\geqslant|a_{12}|+|a_{13}|+|a_{14}|$$
$$|a_{22}|\geqslant|a_{21}|+|a_{23}|+|a_{24}|$$
$$|a_{33}|\geqslant|a_{31}|+|a_{32}|+|a_{34}|$$
$$|a_{44}|\geqslant|a_{41}|+|a_{42}|+|a_{43}|$$

Могут быть получены точные значения X. (Здесь символ $|a|$ обозначает абсолютное значение a; абсолютное значение — это значение, не зависящее от знака.) Заметим, что по крайней мере в одном из приведенных четырех условий должно выполняться строгое неравенство[1].

Напишите программу для решения системы из 4 уравнений методом последовательной аппроксимации. Сначала проведите проверку системы на сходимость. Если система не сходится, напечатайте сообщение SYSTEM WILL NOT CONVERGE и закончите выполнение программы. Процесс решения продолжается до тех пор, пока разность между двумя последовательными аппроксимациями для каждого X не будет меньше, чем заданное число E. В программе считайте E = 0.001. Напечатайте число аппроксимаций, необходимых для получения ответа с заданной точностью.

6. Одной из классических задач является задача нахождения самого короткого пути для посещения коммивояжером всех нужных ему городов. Для решения этой задачи можно воспользоваться методом случайных чисел или так называемым методом Монте-Карло. Напишите программу для определения самого короткого пути посещения девяти городов с помощью алгоритма Моите-Карло, который выполняется следующим образом:

а) Воспользуйтесь генератором случайных чисел вида $RN_{i+1}=RN_i\cdot A$, где $A=237$, RN — случайное число, $RN_1=214748364$. Выделяя 9 цифр младших разрядов RN, определите контрольный путь. В контрольном пути цифры 1—9 должны встречаться только один раз. Следовательно, для точного определения контрольного пути может потребоваться несколько случайных чисел. Например, если при генерировании контрольного пути получается случайное число 481559803, то цепочка контрольного пути будет 481593 или состоять только из 6 городов. Потребуется по крайней мере еще одно случайное число, скажем 176980927. Это второе число даст цифры 762, что дополнит контрольный путь до цифр 48159372.

б) Определите расстояния между городами по треугольной матрице, приведенной далее. Согласно этой матрице город 1 находится в 55 милях от города 9, а город 3 — в 150 милях от города 6. Расстояние контрольного пути 481593762, полученного в шаге 1, равно 85 + 35 + 95 + 80 + 150 + 140 + + 20 + 120, или 825 миль.

---

[1] Система должна быть также «неприводимой», т. е. одно уравнение или более не может быть получено из другого. Для того чтобы решить задачу, в данном примере допустите, что система является неприводимой.

РАССТОЯНИЕ МЕЖДУ ГОРОДАМИ, МИЛЬ

	2	3	4	5	6	7	8	9
1	75	105	95	95	60	50	35	55
2		40	70	140	120	110	95	115
3			80	170	150	140	125	150
4				110	110	105	85	120
5					50	65	60	80
6						20	25	30
7							20	10
8								35

7. Другим способом помочь коммивояжеру можно с помощью метода приращений. Его применение сводится к следующему. Выберите город, просканируйте матрицу расстояний в милях и выберите самый близкий к первому город. Затем выберите самый близкий город ко второму и т. д. Это даст один путь. Затем начните этот процесс с какого-нибудь другого города. Повторяйте этот процесс, выбирая каждый город в качестве отправной точки. В конце концов, выберите самый короткий путь.

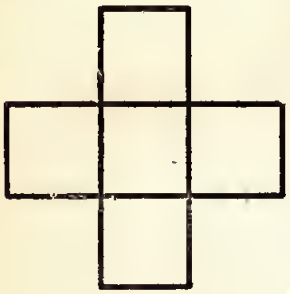


Рис. 4.2.

Напишите программу для нахождения самого короткого пути между девятью городами, данными в задаче 6, с помощью метода приращений. Напечатайте пути и расстояния для каждой из девяти проб и самый короткий из этих девяти путей.

В литературе можно найти описание еще нескольких алгоритмов для решения этой задачи.

8. Механический метод кодирования и декодирования сообщений основан на транслитерации сообщения в заданном порядке. Напишите программу, которая кодирует или декодирует сообщения из 40 символов в зависимости от описателя. Описатель CODE используется для кодирования, а описатель CLEAR — для декодирования, как это показано далее:

а) Сообщение представляет собой матрицу из 8 строк и 5 столбцов. Например, сообщение «Эта программа должна быть написана на языке ПЛ/1» («THIS PROGRAM IS TO BE WRITTEN IN THE PL/1 LANGUAGE») будет иметь вид:

		Столбец				
		1	2	3	4	5
	1	T	H	I	S	P
	2	R	O	G	R	A
	3	M	1	S	T	O
Ряд	4	B	E	W	R	I
	5	T	T	E	N	I
	6	N	T	H	E	P
	7	L	1	L	A	N
	8	G	U	A	G	E

б) Кодирование производится путем перемены местами строк и столбцов способом, о котором условливаются отправитель и получатель. Примером простого правила замены является перемена местами столбцов 5 и 1 и строк 2 и 8. В результате получится следующее декодированное сообщение:

PHISTEVAGGOISTMIEWRBITENTPTHENNILALAOGRR

Исходные данные для описанного выше примера перфорируются на карты в виде:

'CODE' 'THISPROGRAMISTOBEWRITTENINTHEPLILANGU AGE'

После считывания этой строки символов примените функцию SUBSTR для присвоения значения отдельных символов элементам массива. (Существует

более легкий способ присвоения символов, но он будет рассмотрен в одной из следующих глав.)

9. При размещении цифр в квадратах, как это показано на рис. 4.2, цифры от 1 до 8 могут быть расположены: 8! = 8 · 7 · 6 · 5 · 4 · 3 · 2 · 1 = 40 320 различными способами. Напишите программу, которая напечатает все те способы расположения цифр, при которых не будет последовательных чисел в смежных квадратах, по горизонтали, по вертикали или по диагонали.

10. День недели для данной даты можно вычислить с помощью формулы

DAY__OF__WEEK= ([2.6 M−O.2]+D+Y+[Y/4]+[C/4]−2C) mod 7

Дни недели обозначаются цифрами: воскресенье — 0, понедельник — 1 и т. д. M обозначает номер месяца. Январь и февраль считаются 11-м и 12-м месяцем предыдущего года, март — 1-м месяцем, ..., декабрь — 10-м месяцем. D показывает день месяца, Y — год определенного столетия, C — столетие. Квадратные скобки означают целую часть числа. Например, [7.4] = 7 и [7] = 7. Выражение mod 7 показывает, что необходимо вычислить число в круглых скобках, разделить его на 7 и остаток хранить как результат. Для иллюстрации возьмем дату 4 июля 1776 г.:

M=5, D=4, Y=76, C=17
[2. 6M−0.2]=[12.8]=12
[Y/4]=[19]=[19]
[C/4]=[4.25]=4
([2.6M−0.2]+D+Y+[Y/4]+[C/4]−2C) mod 7=(12+4+76+19+4−34) mod7=(81) mod 7=4

Таким образом, 4 июля 1776 г. падает на четверг. Даты перфорируются на карты исходных данных. Они начинаются и заканчиваются апострофом (так, чтобы их можно было считывать как строку символов), и между ними должен быть хотя бы один пробел. Месяц перфорируется полностью, за ним идет один пробел, затем — день месяца, запятая и год. Напишите программу для считывания этих данных и печати каждой даты и дня недели, на который эта дата падает. (Обратите внимание на то, что в ПЛ/1 существуют встроенные функции, которые могут понадобиться при написании программы.)

Глава 5

ВВОД И ВЫВОД ДАННЫХ

5.1. ВВЕДЕНИЕ

Один из наиболее важных аспектов языков высокого уровня заключается в их способности легко обрабатывать различные формы данных. ПЛ/1 обеспечивает передачу данных двумя способами: непрерывным потоком и отдельными записями. В этой главе будет рассмотрена только обработка информации, поступающей в виде непрерывного потока. При этом способе обработки данные рассматриваются как непрерывный поток символов.

Преобразование элементов данных из внешнего представления во внутреннее и из внутреннего во внешнее выполняется автоматически в процессе перезаписи информации.

Передача информации непрерывным потоком осуществляется только двумя операторами: GET и PUT. Оператор GET отыскивает очередные элементы в потоке исходных данных, а оператор PUT направляет элементы в поток выходных данных. Существуют три типа ввода-вывода данных непрерывным потоком: управляемый списком, управляемый данными и управляемый редактированием. Все эти типы будут рассмотрены в следующих разделах.

В настоящей главе предполагается, что для ввода данных используется стандартное системное устройство ввода (SYSIN), которым обычно является перфоратор, а для вывода данных — стандартное системное устройство вывода (SYSPRINT), которое обычно представляет собой построчно печатающее устройство. Один из методов преобразования данных к внутреннему представлению будет рассмотрен в параграфе 5.6.

Для спецификации элементов данных, которые нужно обрабатывать, используется список данных. Он необходим в случае ввода-вывода, управляемого списком, и ввода-вывода, управляемого редак-

тированием, и необязателен в случае ввода-вывода, управляемого данными. Список заключается в круглые скобки, а отдельные элементы данных отделяются друг от друга запятыми. Функции элементов данных зависят от того, предназначен ли список данных для ввода или для вывода. В следующих разделах настоящей главы такой список будет рассматриваться в связи с обсуждением различных способов обработки информации.

5.2. ВВОД-ВЫВОД ДАННЫХ, УПРАВЛЯЕМЫЙ СПИСКОМ

До сих пор мы рассматривали именно такую форму ввода-вывода. Для полноты картины кратко рассмотрим правила, существующие для нее. В предшествующих главах было приведено много примеров такого рода ввода-вывода.

Обычный формат ввода данных имеет вид:

```
[метка:···] GET LIST (список данных) [COPY];
```

обычный формат вывода данных имеет вид:

```
[метка:···] PUT [ PAGE [LINE (w)] ]
                [ SKIP [(w)]      ] LIST (список данных);
                [ LINE (w)        ]
```

В параграфе 2.7 рассматривались дополнительные возможности применения команд PAGE, SKIP и LINE. Конструкция COPY обеспечивает печать потока вводимой информации в том виде, в каком она перезаписывается с одного устройства на другое, и если вывод осуществляется построчным печатающим устройством, то каждый элемент данных будет отпечатан на отдельной строке в той форме, в которой он считан. Особенно ценной эта возможность оказывается при отладке программ, так как контрольные данные и результаты печатаются одновременно.

Список входных данных. Элементы данных в потоке ввода могут быть следующими: [+|—] — арифметическая константа, [+|—] — вещественная константа, {+|—} — мнимая константа, константа, состоящая из строки символов, и константа, состоящая из строки битов. В комплексных числах перед мнимой константой до знака плюс или минус может быть пробел. По крайней мере один пробел и/или запятая отделяют элементы данных друг от друга. Знаки повторения запрещены. Строковые константы заключаются в кавычки, а после констант, состоящих из строк битов, ставится буква B. (В подмножестве ПЛ/1 употребление комплексных констант не допускается.)

Элементы списка данных могут быть переменными, массивами переменных, псевдопеременными или спецификациями повторения (при применении оператора DO), в которые могут входить все виды элементов. Следует помнить, что каждый обязательно заключается в круглые скобки. Если для массива переменных нет спецификации повторения, то они передаются в виде — —, причем самый правый индекс изменяется быстрее всех остальных.

Если требуется оставить переменную в списке данных без изменения при выполнении определенного оператора GET LIST, то соответствующий элемент данных заменяется одним или несколькими пробелами, начало и конец которых задается запятой. Например, если данные отперфорированы на карту таким образом:

```
1, 2, 3, 4, b, 5 (b означает пробел)
```

и выполняется следующий сегмент программы:

```
DO I=1, 2;
GET LIST (A, B, C);
   :
END;
```

то при первом проходе через цикл A, B и C будут читаться соответственно как 1, 2 и 3. При втором проходе A будет присвоено значение 4, текущее значение B останется без изменений, а C будет присвоено значение 5.

Выполнение данного оператора GET LIST заканчивается, когда список данных исчерпывается, или по условию END OF FILE (конец файла). Если условия END OF FILE нет, то очередная команда ввода начнет сканирование вводимого потока данных с позиции, следующей сразу же за пробелом или запятой, которыми закончился последний элемент обрабатываемых данных.

Список выходных данных. Список выходных данных состоит из переменных, массива переменных, псевдопеременных, выражений в виде элементов данных, выражений типа массива, констант или спецификаций повторения перечисленных элементов. Если в списке данных встречается выражение, то сначала оно вычисляется, а затем результат вычисления выдается в поток выходных данных.

Все элементы потока выходных данных разделены пробелами. Во время печати они располагаются на соответствующих стандартных местах. (Для транслятора уровня F системы IBM — это положения 1, 25, 49, 73, 97 и 121). Но программист в своей программе может изменить эти позиции с помощью соответствующих управляющих карт.

Вид элементов данных в выходном потоке зависит от их описателей. Двоичные данные преобразуются в десятичные. Строки битов заключаются в кавычки и в конце строки стоит буква B. Строки символов печатаются в кавычках.

5.3. ВВОД-ВЫВОД, УПРАВЛЯЕМЫЙ ДАННЫМИ

В подмножестве ПЛ/1 ввод-вывод, управляемый данными, не употребляется.

Метод ввода-вывода, управляемого данными, во многих отношениях проще метода ввода-вывода, управляемого списком, так как при необходимости список данных может быть опущен. Кроме того, этот метод часто более удобен для проверки программы, так как выход представляется в форме оператора присваивания, а имя переменной и ее значение печатаются.

Общая форма ввода, управляемого данными, имеет вид:

```
[метка:···] GET DATA [(список данных)] [COPY];
```

Общая форма вывода, управляемого данными, имеет следующий вид:

```
[метка:···] PUT [PAGE [LINE (w)]
                SKIP [(w)]
                LINE (w)      ] DATA [(список данных)];
```

Конструкции PAGE, SKIP, LINE и COPY служат для тех же целей, что и при вводе-выводе, управляемом списком.

Оператор GET DATA. Элементы данных отделяются друг от друга запятой и/или одним или несколькими пробелами. В операторе GET конец набора входных данных задается точкой с запятой. Значения элементов должны иметь форму правильно определенных констант. Список данных необязателен. Имена переменных в потоке входных данных определяют, какие значения считываются, а положение точки с запятой определяет, сколько элементов передается после одного оператора GET. При желании может быть составлен и список переменных. Обычно в него включаются имена переменных с тем, чтобы знать, какие данные считываются тем или иным оператором. Порядок размещения имен переменных в списке данных может не совпадать с порядком, в котором они появляются в потоке входных данных, и, более того, появление каждой переменной списка в потоке входных данных необязательно. Однако если переменная и ее значение появляются в потоке входных данных, а в программе, которая определяет список данных, имя этой переменной отсутствует, то печатается сообщение об ошибке и этот элемент игнорируется.

В качестве примера этого типа ввода данных рассмотрим сегмент программы:

```
DO I=1 TO 3;
GET DATA;
PUT LIST (A+B+C+D);
END;
```

Предположим, что данные отперфорированы на карте следующим образом:

```
A=1, C=2, B=3, D=4; B=10; C=7, A=6;
```

При первом проходе цикла значения A, B, C и D соответственно равны 1, 2, 3 и 4, их сумма вычисляется и печатается. При втором проходе цикла значение B изменится на 10 и сумма 1 + 10 + 2 + 4 будет вычислена и напечатана. При третьем (последнем) проходе цикла значения C изменятся на 7, а A — на 6 и будет вычислена и напечатана сумма 6 + 10 + 7 + 4.

В ранее приведенном примере оператор GET DATA (A, B, C, D); мог бы служить переменной. Однако запись GET DATA (A, B, C); вызвала бы сообщение об ошибке, так как переменная D имеется во входных данных.

Список данных, если он необходим, должен состоять только из имен переменных без индексов или имен массивов. Индексированные

переменные могут использоваться во входных данных. Например, пусть A объявлено массивом 10 × 10. Для чтения данных A (1, 2) = = 3, A (8, 7) = 2; применима либо команда GET DATA; либо команда GET DATA (A); после выполнения оператора GET изменятся только значения A (1, 2) и A (8, 7). Такой способ полезен, когда нужно прочитать только несколько значений из большого массива.

Оператор PUT DATA. Способ, которым определяется текущее значение переменной, задается описателями, которые ей приписываются. Эти значения имеют ту же форму, что и при выводе данных, управляемом списком. Переменные выводятся потоком, элементы разделяются пробелами, а за списком для оператора PUT DATA следует точка с запятой. Место печатаемых выходных данных определяется так же, как в операторе LIST при печати списка. Строки печатаются в кавычках, а после строки битов ставится буква B.

Простой оператор PUT DATA обеспечивает вывод всех переменных, известных к моменту выполнения оператора.

Список данных необходим только в том случае, когда требуется напечатать некоторые значения переменных. Порядок их печати определяется списком данных. Элементы списка данных при выводе могут состоять из переменной (с индексами или без них) или массива. В списке данных для вывода разрешены повторные спецификации при использовании операторов DO. Следует обратить внимание на то, что в отличие от вывода, управляемого списком, в выводе, управляемом данными, выражения и константы не применимы.

5.4. ВВОД-ВЫВОД, УПРАВЛЯЕМЫЙ РЕДАКТИРОВАНИЕМ

Из всех рассмотренных ранее операторов ПЛ/1 операторы ввода-вывода, управляемого редактированием, больше всего похожи на соответствующие операторы Фортрана. При передаче данных в непрерывном потоке программист управляет форматом элементов данных в этом потоке. По существу, операторы ПЛ/1 в этом случае аналогичны операторам Фортрана READ и WRITE с соответствующим оператором FORMAT. Если читатель знаком с оператором и спецификациями FORMAT в Фортране, то ему относительно легко овладеть материалом, изложенным в этом параграфе.

Существуют два основных различия в подходе к формату в ПЛ/1 и в Фортране. Первое заключается в том, что в ПЛ/1 спецификации формата либо могут включаться в команду ввода-вывода, либо, как в Фортране, могут употребляться в виде отдельного оператора. Второе различие, которое, пожалуй, более существенно, состоит в том, что спецификация формата для вводимого элемента данных относится только к элементу, находящемуся на внешнем носителе. Преобразование из внешней формы во внутреннюю, определяемое описателями переменной, производится автоматически. В противоположность этому подходу спецификация формата в Фортране предполагает преобразование определенных внутренней и внешней форм. В результате такого ограничения при применении Фортрана программист вынужден согласовывать тип формата, который он употребляет, как с типом перемен-

ной, так и с форматом выходных данных. Например, спецификация F в Фортране предполагает преобразование внешней формы числа с фиксированной точкой во внутреннюю форму с плавающей точкой. Спецификация формата F в ПЛ/1 предполагает преобразование любой внутренней формы во внешнюю форму с фиксированной точкой.

Существует еще несколько мелких различий, о которых будет упомянуто в соответствующих параграфах.

Общая форма оператора ввода, управляемого редактированием, имеет вид:

```
[метка:···] GET EDIT (список данных) (формат)
[(список данных) (формат)] ··· [COPY];
```

А общая форма оператора вывода, управляемого редактированием, следующая:

```
                  ┌PAGE [LINE (w)]┐
[метка:···] PUT   │SKIP [(w)]     │ EDIT (список данных)
                  └LINE (w)       ┘
(формат) [(список данных) (формат)]···;
```

Конструкции PAGE, SKIP, LINE и COPY при этом способе ввода-вывода служат тем же целям, что и при вводе-выводе, управляемом списком.

Списки данных. Список входных данных может состоять из тех же самых элементов, разделенных запятыми, что и при вводе, управляемом списком, а именно: переменных, массива переменных, псевдопеременных или повторных спецификаций (при использовании оператора DO), в которые могут включаться любые из перечисленных элементов. Список данных и каждый оператор DO обязательно заключаются в круглые скобки:

```
(M, ((A (I, J) DO I=1 TO M BY 2) DO J=1 TO 6), X, Y, Z)
```

Порядок ввода входных данных в потоке определяется форматом, связанным со списком входных данных, а не с описателями переменных.

Список выходных данных может состоять из переменной, массива переменных, псевдопеременной, выражения, выражения типа массив, константы или повторной спецификации, состоящей из всех этих элементов. Если в списке данных встречается выражение, то оно сначала вычисляется, а затем вводится в поток выходных данных, форма выходных данных определяется спецификацией связанного с ними формата.

Если в списке данных встречается массив, то все его элементы в соответствии со спецификациями формата располагаются в порядке номеров строк, т. е. с самым быстрым изменением правого индекса.

Форма данных в непрерывном потоке. Как при вводе, так и при выводе данных, элементы в потоке располагаются в зависимости от связанной с ними спецификации формата. В отличие от ввода-вывода, управляемого списком, и ввода-вывода, управляемого данными, эти элементы не разделяются автоматически друг от друга пробелом или специальным символом. Список данных определяет значения обрабаты-

ваемых данных, а список формата определяет порядок ввода-вывода этих значений в потоке.

При выводе данных строки не заключаются в кавычки и после строк битов буква В не ставится. При вводе строк кавычки и буква В тоже не употребляются.

Строки при выводе потока данных левоустановленные, а арифметические данные правоустановленные. Если спецификация ширины поля мала, то строки данных усекаются справа, а арифметические данные — слева.

Двоичные фиксированные константы и константы с плавающей точкой преобразуются при выводе в десятичные и во входном потоке должны иметь форму десятичных констант.

Каждый элемент не печатается в каком-либо заранее определенном месте на печатающем устройстве и даже выполнение нового оператора PUT не приводит к тому, чтобы печать начиналась с новой строки. Элементы данных печатаются непрерывно один за другим, вдоль всей строки до полного ее заполнения. Затем печатающее устройство автоматически передвигает бумагу на новую строку. Число символов, которое может быть напечатано на каждой строке, и число строк на странице зависят от устройства ввода. Программист может изменить их с помощью соответствующих управляющих карт. Элементы управления форматом служат для прерывания непрерывного процесса печатания и получения желаемого размещения знаков.

Формат. Формат в операторах GET или PUT бывает двух типов. Он может состоять из действительного списка элементов формата, разделенных запятыми. Он может также отсылать к метке, которая определяет оператор программы, содержащий действительный список форматов. Второй тип очень похож на оператор FORMAT в Фортране и называется в ПЛ/1 удаленным форматом (R-формат).

В любом из этих двух случаев формат заключается в круглые скобки. Удаленный формат обычно имеет вид:

```
(R (метка) )
```

где метка может быть либо константой типа метки, либо переменной типа метки (ей приписывается постоянная величина), которая отсылает к R-формату. Спецификация R-формата имеет следующий вид: метка 1: [метка 2: ...] FORMAT (список спецификаций формата); она может находиться в любом месте блока, содержащего операторы GET или PUT. Оператор FORMAT пропускается при выполнении нормальной последовательности программы, а применяется только операторами GET или PUT, в которых метка представляет собой метку R-формата.

Следующие два примера иллюстрируют эти два различных типа формата. Они эквивалентны.

```
        PUT EDIT (I, X) (F (5), E (15,5) );
        PUT EDIT (I, X) (R (ABC) );
ABC:    FORMAT (F (5), E (15, 5) );
```

Отдельные элементы формата делятся на элементы формата данных и элементы управления форматом. При вводе или выводе информации

первый элемент в списке данных определяется элементом формата в списке форматов, второй элемент — вторым элементом формата и т. д. Если перед следующим элементом формата имеются какие-нибудь элементы управления форматом, то они выполняются до того, как значение данных попадает в поток. Выполнение операторов GET или PUT заканчивается, как только закончится ввод-вывод соответствующих элементов данных в потоке, даже если в списке еще остаются элементы форматов, включая и элементы управления форматом.

Если элементов данных больше, чем описано элементами формата, то список элементов формата повторяется с самого начала. Заметьте, что такой порядок несколько отличается от правил Фортрана. Чтобы проиллюстрировать это, рассмотрим такой R-формат:

```
XYZ:  FORMAT (SKIP, E (20, 5), E (15, 5), SKIP (2) );
```

Оператор PUT EDIT (A) (R (XYZ)); обеспечит печать со следующей строки и напечатает на ней текущее значение A в соответствии с форматом E (20, 5).

Оператор PUT EDIT (A, B) (R (XYZ)); обеспечит печать с новой строки и в соответствии с форматом E (20, 5) напечатает на ней текущее значение A, а в соответствии с форматом E (15, 5) — текущее значение B на этой же строке. Элемент формата SKIP (2) выполняться не будет, так как список данных исчерпан.

Оператор PUT EDIT (A, B, C) (R (XYZ)); обеспечит печать с новой строки. На ней согласно формату E (20, 5) будет напечатано текущее значение A и согласно формату E (15,5) — текущее значение B. Поскольку элемент C еще остается в списке, будет выполняться спецификация SKIP (2), что приведет к пропуску двух строчек. В этом случае список формата исчерпан и его выполнение начинается снова. Спецификация SKIP приводит к пропуску еще одной строки и затем печатается текущее значение C в соответствии с форматом E (20, 5).

Следует всегда помнить, что если строка полностью не заполняется после выполнения одного оператора PUT, то печатающее устройство автоматически не передвигает бумагу на новую строку при выполнении следующего оператора PUT. Программист с помощью спецификаций управления форматом обеспечивает вывод данных необходимым образом.

Управление элементами формата. Элементы управления форматом не оказывают никакого эффекта на выполнение программы, если только они не встречаются раньше, чем исчерпывается список данных. В следующих описаниях предполагается выдача на печать и w может быть константой или выражением. Последнее вычисляется и преобразуется в целое число во время выполнения программы.

Оператор PAGE вызывает прогон бумаги до первой печатаемой строки на следующей странице.

Оператор SKIP [(w)] обеспечивает пропуск w строк, начиная от текущей строки. Следует отметить, что пропускается w — 1 строчка. Если число w опущено, то предполагается, что оно равно единице. Если w меньше или равно 0, то происходит возврат каретки без пропуска строки.

Если значение w настолько велико, что оно должно вызвать пропуск числа строк, которое больше оставшегося на странице, то возникает условие ENDPAGE (которое будет рассмотрено далее), и по системе умолчания первая строка новой страницы пропускается, а SKIP, если он записан, игнорируется. (В подмножестве ПЛ/1 число w может быть только константой со значениями 0, 1, 2, 3.)

Оператор LINE(w) обеспечивает передвижение бумаги на строку w текущей страницы. Заметьте, что эта команда определяет абсолютный номер строки на странице, в то время как оператор SKIP — относительное положение от текущей строки. Если строка, определяемая числом w, уже пройдена или если размер страницы не допускает печать, возникает условие ENDPAGE, и по умолчанию все строки данной страницы пропускаются, а бумага передвигается к первой строке новой страницы. Вариант LINE игнорируется[1].

Оператор X (w) определяет, что при выводе должно быть добавлено w пробелов или пропущено w символов. Если число w меньше 0, то оно принимается равным 0[1].

Оператор COLUMN (w) определяет, что следующее поле начинается со столбца w текущей строки. Если столбец, определяемый числом w, находится за пределами размера строки или меньше единицы, то его значение принимается равным единице. Обратите внимание на то, что оператор COLUMN (w) определяет абсолютное положение на текущей строке, в то время как оператор X (w) — относительное положение.

Оператор COLUMN (w) определяет колонку карты, с которой начинается ввод[1].

Элементы формата данных. Символы w, d, s и p могут быть константами или выражениями. Выражения вычисляются и преобразуются в целые числа во время выполнения программы. (В подмножестве ПЛ/1 w, d и s должны быть десятичными целыми константами без знака, p — десятичной целой константой со знаком. w имеет следующие ограничения: для форматов A и X $w < 256$, для формата B $w < 65$, а для форматов E и F $w < 33$.)

E-формат с плавающей точкой. Общий тип такого формата E (w, d [s]), где w — ширина поля в потоке ввода-вывода, d — число цифр, которые следуют за десятичной точкой, s — число цифр в мантиссе.

При выводе данных это число является правоустановленным в поле шириной w и имеет следующую форму:

[—] {s — d цифры}.{d цифры} E {+|—} показатель степени из 2 цифр

Показатель степени определяется так, чтобы старшая значимая цифра в мантиссе не была равна нулю. Если число s не определено, то оно принимается равным 1 + d. Если внутренняя форма представления данных преобразуется в число с плавающей точкой, то в том случае, когда усечение приводит к потере цифры справа, производится округление.

[1] В подмножестве ПЛ/1 число w должно быть десятичной целой константой без знака, меньшей или равной 256.

Программист должен определять ширину поля w, достаточную для помещения в поле знака (если он минус), десятичной точки, мантиссы, E и показателя степени, состоящего из двух цифр со знаком.

При выводе данных внешняя форма представления с плавающей точкой следующая:

[+|−]мантисса [{[E] {+|−}} показатель степени]
{ E {+|−} }

где показатель степени — десятичная целая константа, состоящая из двух цифр, а мантисса — десятичная константа с фиксированной точкой. Если показатель степени опущен, то он принимается равным 0. Число может быть занесено на любое место поля с шириной w. Если десятичная точка задана, то она, а не значение d, определяет число цифр, следующих за десятичной точкой. Иными словами, d задает число знаков в мантиссе справа от предполагаемой десятичной точки.

Примеры (b означает пробел)

Внутреннее представление	Формат	Внешнее представление
+1234.567	E (15, 7)	bb1.2345670E+03
−1234.567	E (15, 7)	b−1.2345670E+03
+1234.567	E (13, 5)	bb1.23457E+03
−1234.567	E (13, 5)	b−1.23457E+03
+1234.567	E (13, 5, 7)	b12.34567E+02
−1234.567	E (13, 5, 7)	−12.34567E+02
+1234.567	E (13, 2, 7)	b12345.67E−01
−1234.567	E (13, 2, 7)	−12345.67E−01
+1234.567	E (10,7)	ошибка, ширина поля недостаточна

F-формат с фиксированной точкой. Общая форма такого формата имеет вид: F (w [, d [, p]]), где w задает ширину поля, d — число цифр, которые следуют за десятичной точкой, p — масштабный коэффициент, и, возможно, знак плюс или минус.

При выводе данных это число является правоустановленным в поле шириной w. Если число w задано, то выполняется только его целая часть, без десятичной точки. Если заданы w и d, то десятичная точка помещается перед d правоустановленными цифрами. Если задано p, то обрабатываемая величина в 10^p раз больше величины во внутреннем представлении.

При вводе данных число может быть в любом месте поля шириной w. Если d не задано, то предполагается, что оно равно 0. Если d задано, то оно определяет число цифр справа от предполагаемой точки. С другой стороны, если точка занесена в поле, то она, а не d, определяет число цифр после точки. Если задан масштабный коэффициент p, то он умножает данные во внешнем представлении на 10^p.

Примеры (b означает пробел)

Внутреннее представление	Формат	Внешнее предстаааленне
+1234.567	F (4)	1235
−1234.567	F (4)	1235
+1234.567	F (9)	bbbbb1235
−1234.567	F (9)	bbbb−1235
+1234.567	F (9, 3)	b1234,567
−1234.567	F (12, 2)	bbbb−1234.57
+1234.567	F (12,4)	bbb1234.5670
−1234.567	F (12, 3, 2)	b−123456.700
+1234,567	F (3)	235

С-формат комплексных переменных. (В подмножестве ПЛ/1 употребление этого формата запрещено.) Общий тип С-формата:

С (вещественное ['вещественное])

Если вторая спецификация «вещественное» опущена, то ее предполагают равной первой. Две спецификации «вещественное» определяют обработку вещественной и мнимой частей комплексного числа. Вещественными спецификациями могут быть E, F или P. Мнимое I как при вводе данных, так и при выводе их не используется. Знак перед мнимой частью печатается только в том случае, когда ее значение меньше 0 или когда она определяется спецификацией P.

P-формат шаблона. Этот вид спецификации формата подробно будет рассмотрен в главе 7; здесь он приводится только для сведения.

A-формат строки символов. Общий тип такого формата: A [(w)].

При выводе спецификации A обеспечивает преобразование элемента списка данных в строку символов длиной w. Эта константа левоустановленная. Она не заключается в кавычки. Если число w не задано, то оно предполагается равным ширине поля элемента в списке данных после необходимых преобразований.

При вводе определение w требуется всегда. Оно показывает, что следующие w символов должны быть преобразованы в строку символов и помещены в поток выходных данных.

Пример (b означает пробел)

Внутренняя форма	Формат	Внешняя форма
'ABC'	A	ABC
'ABC'	A(3)	ABC
'ABC'	A(5)	ABCbb
'ABC'	A(2)	AB

B-формат строки битов. Общий тип имеет вид: B [(w)]

Эта спецификация предназначена для обработки строк битов и выполняется точно так же, как спецификация A. Кроме того, при вводе первые и последние пробелы игнорируются, а при выводе w обязательно должно быть задано, если только соответствующий элемент списка данных не представляет собой строку битов. Буква B и кавычки никогда не употребляются при выводе, а при вводе они могут быть опущены.

Примеры (b означает пробел)

Внутреннее представление	Формат	Внешнее представление
'11111'B	B	11111
'11111'B	B(5)	11111
'11111'B	B(7)	11111bb
'11111'B	B(3)	111

5.5. ДОПОЛНИТЕЛЬНЫЕ СВЕДЕНИЯ О ВВОДЕ-ВЫВОДЕ, УПРАВЛЯЕМОМ РЕДАКТИРОВАНИЕМ

Список формата. Список формата, который всегда заключается в круглые скобки, либо следует сразу за списком данных, либо задается отдельным оператором формата. Каждый элемент списка формата

отделяется один от другого запятой. Общий вид списка формата имеет вид:

$$\left\{\begin{array}{l}\text{элемент формата}\\ n\text{ — элемент формата}\\ n\text{ (список формата)}\end{array}\right\} \qquad \left[\begin{array}{l},\ \text{элемент формата}\\ ,\ n\text{ — элемент формата}\\ \quad n\quad\text{(список формата)}\end{array}\right]\ldots$$

В этой общей форме n представляет собой коэффициент повторения и может быть выражением, заключенным в круглые скобки или целой константой без знака. Если n — выражение, то во время выполнения программы оно вычисляется и при необходимости преобразуется в целое число. Если n меньше или равно 0, то определяемый им элемент формата или список формата пропускается. Если n — целое число, то оно должно отделяться от следующего элемента формата пробелом. (В подмножестве ПЛ/1 число n должно быть целой константой без знака.) Приведенные далее примеры иллюстрируют употребление этой конструкции.

Списки элементов формата (F (9, 2), F (9, 2)) и (2F (9, 2)) эквивалентны, так же как (A, F (9, 2), F (7, 2), F (9, 2), F (7, 2)) и (A, 2 (F (9, 2), F (7, 2))).

Список элементов формата ((N) E (120/ N, 7)) может служить для размещения *N* значений на первых 120 печатаемых позициях строки. Нужное значение N вычисляется во время выполнения программы (например, N может иметь значение 5, но не может иметь значения 50). Значение, которое при вводе присваивается переменной, может быть использовано в различных элементах списка формата. Например,

```
GET EDIT (I,STRING, J, X) (F (2), A (I), F (2), F(J) );
```

В этом примере целое число I при вводе указывает длину строки символов STRING. Целое число J обозначает длину целого числа X.

Возможность использования в списке элементов формата выражений и рассмотренных методов записи — очень сильная сторона языка ПЛ/1. С помощью R-формата программист может изменить формат во время выполнения программы, как это показано в следующем примере:

```
        GET LIST (X);
        IF X=1 THEN XYZ=LABEL__A;
               ELSE XYZ=LABEL__B;
        GET EDIT (A, B, C) (R (XYZ) );
LABEL__A: FORMAT (3 F (5) );
LABEL__B: FORMAT (2 F (6, 2), E (13, 5) );
```

В этом примере XYZ как метка должна быть объявлена описателем LABEL.

При выборе элементов формата нужно учитывать, что коэффициент повторения, равный или меньший 0, вызывает пропуск этого элемента.

В следующем операторе

```
PUT EDIT (A, B, C) ( (I) F (6,2), (1—I) F (8,3), (2—I) E (15,7) );
```

значения A, B и C появятся при выводе в соответствии со следующими форматами (в зависимости от текущего значения I):

	A	B	C
I ≤ −2	F (8,3)	F (8,3)	F (8,3)
I = −1	F (8,3)	F (8,3)	E (15,7)
I = 0	F (8,3)	E (15,7)	E (15,7)
I = 1	F (6,2)	E (15,7)	F (6,2)
I ⩾ 2	F (6,2)	F (6,2)	F (6,2)

Смешение трех типов ввода данных в одной программе. Напомним, что ввод данных, управляемый списком или редактированием, заканчивается, когда список данных исчерпывается, а окончание ввода, управляемого данными, определяется точкой с запятой (закончить ввод можно и в результате выполнения условия «конец файла», но сейчас этот способ мы рассматривать не будем). Размещение входных данных для следующего оператора GET зависит от типа только что закончившейся передачи данных.

После выполнения ввода в соответствии с LIST или DATA следующий символ, который передается в соответствующий оператор GET, относится к вводу по типу LIST, если этот символ отделяется от последнего элемента пробелом или запятой. Символ, следующий непосредственно за точкой с запятой, вводится по типу DATA. При вводе, управляемом редактированием, следующим для передачи символом будет тот, который следует непосредственно за последним символом в последнем передаваемом элементе.

При последовательной обработке операторов типа LIST или DATA дополнительные пробелы, стоящие впереди, игнорируются, но если следующий оператор GET управляется редактированием, нужно быть очень осторожным, поскольку в этом случае пропуски не игнорируются.

В качестве примера рассмотрим следующую последовательность операторов GET:

```
GET LIST (A);
GET DATA (B, C);
GET LIST (D);
GET EDIT (E) (F (3) );
GET EDIT (F) (X (2), F (5));
GET LIST (G);
```

Если входными данными являются

```
12.3 bb B = 456.7, C = 0.21; bbb14.2b216bbb2345bb28bb
```

(b означает пропуск), то подчеркнутые символы показывают расположение данных во входном потоке для различных операторов GET.

Эффективное использования формата. Каждый элемент списка формата генерирует программы, проверяющие необходимость преобразования данных. Требования к памяти и время выполнения программы могут сократиться при уменьшении длины списка элементов формата. Например, следующие два оператора дают одинаковый выход:

```
PUT EDIT (A, B, C, D) (A(3), X(4), A(10), X(2), A(5), X(2), A(20));
```

и

```
PUT EDIT (A, B, C, D) (A(7), A(12), A(7), A(20));
```

так же, как и следующие два:

```
PUT EDIT (I, X) (X(4), F(3), X(7), F(5,2));
```

и

```
PUT EDIT (I, X) (F(7), F(12,2));
```

В том и в другом случае второй оператор лучше с точки зрения техники программирования. Подобный метод может применяться и при вводе для уменьшения длины списка элементов формата.

Часто программист бывает вынужден обрабатывать полученные данные без управления внешней формой данных. Рассмотрим такой пример.

На картах исходных данных в первых 25 колонках отперфорированы фамилии рабочих; вслед за фамилиями, в колонках 26 и 27, отперфорированы часы их работы. Необходимо считать эти данные и обработать их. Следующий сегмент программы будет считывать фамилию первого рабочего, часы его работы, обрабатывать эти данные и затем при считывании второго элемента данных выдавать сообщение об ошибке.

```
IN; GET EDIT (ФАМИЛИЯ, ЧАСЫ) (A(25), F(2), X(53));
    .
    .       (операторы обработки)
    .
GO TO IN;
```

Помните, что после того, как список данных исчерпан, X (53) игнорируется.

Целью указанного сегмента был пропуск последних 53 колонок перфокарты. Это можно было выполнить и другими способами, например с помощью любого из следующих операторов GET:

```
GET EDIT (ФАМИЛИЯ, ЧАСЫ) (COLUMN (1), A(25), F(2));
GET EDIT (ФАМИЛИЯ, ЧАСЫ, A) (A(25), F(2),X(52), A(1));
GET EDIT (ФАМИЛИЯ, ЧАСЫ, B) (A(25), F(2), A(53));
```

где A и B представляют собой пустые строки символов длиной 1 и 5 соответственно. Когда требуется прочитать и обработать много перфокарт, последний сегмент более эффективен.

5.6. ОПЕРАТОРЫ GET И PUT STRING

До сих пор операторы GET и PUT рассматривались в связи с передачей данных на внешнее устройство и обратно. В этом разделе операторы GET и PUT будут рассматриваться в связи с внутренним процессом пересылки данных.

Для иллюстрации возьмем простой пример: переменная CARD задана описателем CHARACTER (80).

В CARD хранится строка символов с отперфорированной фамилией в позициях 1—25. Позиции 26 и 27 показывают количество часов рабочего времени, а позиции 28 и 29 — количество часов сверхурочной

работы. Необходимо разместить эти элементы в различных ячейках памяти и обработать их. Оператор

```
GET STRING (CARD) EDIT (ФАМИЛИЯ, РАБОЧИЕ ЧАСЫ,
  СВЕРХУРОЧНЫЕ ЧАСЫ) (A(25), 2F(2));
```

выполнит эту задачу. Оператор STRING показывает внутреннюю пересылку, а CARD — строку символов во внутренней памяти, которая должна сканироваться.

Подобным способом можно воспользоваться оператором PUT для объединения данных во внутренней памяти в строку символов:

```
PUT STRING (ОПЛАТА) EDIT (ФАМИЛИЯ, РАБОЧИЕ_ЧАСЫ* РАСЦЕНКА
+СВЕРХУРОЧНЫЕ_ЧАСЫ* СВЕРХУРОЧНАЯ_РАСЦЕНКА) (A(25), F(15,2));
```

В этом примере у слова «ОПЛАТА» должен быть описатель CHARACTER и оно должно иметь длину по крайней мере 40.

Оператор STRING может служить для обозначения как строк символов, так и строк битов со всеми тремя способами ввода-вывода. Но особенно часто он используется при вводе-выводе, управляемом редактированием, так как при вводе-выводе, управляемом списком или данными, требуется, чтобы элементы данных отделялись друг от друга пробелами. А ввод-вывод, управляемый данными, требует еще наличия точки с запятой, что определяет конец перезаписи и операторов присваивания.

Если набор данных состоит из двух типов перфокарт и на каждой в первой колонке отперфорировано 1 или 2 для указания типа перфокарты, то с помощью следующего сегмента программы можно считывать эти перфокарты (такая операция может быть выполнена и без оператора STRING, как было показано в предыдущем параграфе.)

```
IN;  GET EDIT (ВХОДНЫЕ ДАННЫЕ) (A(80));
     GET STRING (ВХОДНЫЕ ДАННЫЕ) EDIT (CODE) (A(1));
     IF CODE = 1 THEN GET STRING
     (ВХОДНЫЕ ДАННЫЕ) EDIT (X, Y, Z) (X(1),3F(8,2));
                ELSE GET STRING (ВХОДНЫЕ ДАННЫЕ) EDIT (A, B) X(1),  A(9),  F(7));
```

В качестве другого примера рассмотрим набор данных, отперфорированных двумя разными операторами. Первый оператор перфорирует фамилию в колонках 1—25, часы рабочего времени — в колонках 26—27, часы сверхурочной работы — в колонках 28—29. Второй оператор перфорирует фамилию в колонках 1—30, часы рабочего времени — в следующих двух колонках, а сверхурочные часы работы — в следующих за ними двух колонках. Все эти карты смешиваются в одну общую колоду. Один из способов прочитать эти пефокарты состоит в перфорировании кода 1 или 2 в колонке 80. Код показывает тип карты. Затем карты считываются так, как это иллюстрирует следующий пример:

```
IN;   GET EDIT (ВХОДНЫЕ ДАННЫЕ) (A(80));
      GET STRING (ВХОДНЫЕ ДАННЫЕ) EDIT (CODE) (X(79), A(1));
      IF CODE = 1 THEN ABC=FMT1;
                  ELSE ABC=FMT2;
      GET STRING (ВХОДНЫЕ ДАННЫЕ) EDIT (ФАМИЛИЯ, РАБОЧИЕ_ЧАСЫ СВЕРХУ-
        РОЧНЫЕ_ЧАСЫ) (R (ABC));
```

```
FMT1: FORMAT (A(25), 2F(2));
FMT2; FORMAT (A(30), 2F(2));
```

Этот метод перфорации требует специального кода в конце каждой карты. Проверку такого типа можно производить на любой заранее определенной колонке карты. В приведенном примере наличие перфорации в колонке 31 указывает на различие двух типов карт.

5.7. ОПЕРАТОРЫ DISPLAY — REPLY

Оператор DISPLAY обеспечивает воспроизведение сообщения на пульте оператора вычислительной машины. Ответ человека-оператора машина может потребовать с помощью оператора REPLY.

Общая форма этих операторов имеет вид:

```
[метка: ...] DISPLAY (выражение) [REPLY (символьная переменная)];
```

При выполнении этих операторов «выражение» вычисляется, преобразуется в строку символов (если это необходимо) и затем воспроизводится на пульте. Максимальная длина этой строки зависит от применяемой аппаратуры. Обычно она достигает примерно 100 символов. Если ответ не требуется, то выполнение программы продолжается.

Если же требуется ответ, то после выдачи сообщения выполнение программы прерывается и возобновляется вновь после ответа оператора в виде строки символов. Строка символов, которой отвечает оператор, становится значением «символьной переменной» и может использоваться в программе.

Примеры

```
DISPLAY ('КОНЕЦ РАБОТЫ');
DISPLAY ('КОНЕЦ ШАГА' ||N);
DISPLAY ('ПОЖАЛУЙСТА РАСПЕЧАТКУ') REPLY (ИМЯ ОПЕРАТОРА);
DISPLAY ('НЕТ СХОДИМОСТИ ПОСЛЕ' ||N||' ЦИКЛОВ. СЛЕДУЕТ ЛИ УМЕНЬШИТЬ НЕ-
  ВЯЗКУ НА АХ10? ОТВЕТЬТЕ «ДА» ИЛИ «НЕТ»)
REPLY (ОТВЕТ);
```

В двух последних примерах для получения ожидаемого ответа «ИМЯ ОПЕРАТОРА» и «ОТВЕТ» следует объявлять строками символов подходящей длины. В последнем примере ответ можно проверить с помощью оператора IF. После этого будут выполнены необходимые операции.

Эти операторы следует применять только при абсолютной необходимости, так как они могут резко уменьшить производительность процессора, к тому же оператор REPLY требует непосредственного вмешательства человека. Современные большие вычислительные системы и так слишком загружают операторов, и программистам не следует добавлять им работы. Оператор DISPLAY целесообразен только для сообщения необходимой информации человеку-оператору: записи операций, проведенных в его отсутствие, или сообщений о возникших ненормальных ситуациях во время выполнения программы.

5.8. УПРАЖНЕНИЯ

Короткие упражнения

1. Если форма внутреннего представления переменной —764.87, то какова будет форма ее представления на внешнем устройстве прн следующнх форматах?

```
F(8, 3)        E(10, 3)
F(8)           E(10, 2)
F(9, 3, 1)     E(9, 3)
F(9, 3, —1)    E(10, 3, 4)
F(8, 1)        E(12,5)
F(8, 0, 1)     E(15, 2, 5)
```

2. Массивы A, B н C одномерные, каждый массив содержнт 64 элемента. В этнх массивах хранятся вычисленные значения, которые можно напечатать без потери точностн с помощью формата F (6,2). Напншите однн оператор PUT EDIT, который:

а) напечатает элементы массива A по одному чнслу на строке, с одним интервалом между строк, по 16 чисел на странице;

б) сделает то же самое, что в (а), но только с двумя интерваламн;

в) напечатает элементы массивов A, B н C тремя столбцами, с одним интервалом между строк, 32 строкн на странице;

г) сделает то же, что в прнмере (в), но напечатает заголовкн ARRAY A, ARRAY B н ARRAY C на каждой странице в центре над каждой колонкой. После заголовка две строчкн должны быть пропущены.

3. Массивы A н B одномерные, в каждом 64 элемента. Напншите оператор GET, который будет считывать данные в эти массивы при условии, что:

а) имеются 64 перфокарты н на каждой карте содержится по одному значению для масснва A, которые отперфорнрованы в колонках 1—35, значення для массива B отперфорированы в колонках 40—75. В том и в другом случае чнсла отперфорнрованы с десятичной точкой;

б) значения для массивов A и B отперфорнрованы поочередно в поле шириной 10 на 16 картах. Каждое чнсло отперфорировано с десятичной точкой;

в) отперфорированы значения для целого масснва A, за нимн — значения для массива B. Каждое значение занимает поле шириной 10 на 16 картах. Каждое число отперфорировано с десятичной точкой.

4. Массив A имеет размерность (20,20) н каждому его элементу присвоено начальное значенне 0;

а) значение 12.76 было отперфорировано на карте в колонках 1—5 и был выполнен оператор GET LTST (X); Необходимо считать значения 13.2 и 15.75 и хранить нх в A (5,9) и A (7,18). Напишите оператор GET, который выполнит эти операции, н объяснnте, как нужно отперфорировать данные на одной карте;

б) напишите оператор GET для счнтывання значений в главную диагональ массива A. Объясните способ перфорации карты данных.

5. Напишите один оператор PUT, который эквивалентен следующим сегментам программы:

а)
```
DECLARE A(50);
DO I=1 TO 30;
PUT EDIT (A(I)) (SKIP, F(7, 2));
END;
```

б)
```
DECLARE A(10, 30);
DO I=1 TO 10;
DO J=1 TO 30;
PUT SKIP EDIT (A(I, J)) (F(7, 2));
END; END;
```

в)
```
DECLARE A(10, 30);
DO J=1 TO 30;
DO I=1 TO 10;
PUT SKIP EDIT (A(I, J)) (F(7, 2));
END; END;
```

6. Напишите один оператор PUT, который обеспечит печать строки символов 'ПЛ/1' точно в центре третьей строки иовой страиицы, а также напечатает 'PAGE 1' в крайней правой части этой строки. Пусть размер строки равен 120.

7. Массив A объявлеи FIXED (10, 4) с размериостью (5, 600). После вычислеиия все эти зиачеиия необходимо иапечатать. Напишите одии оператор PUT, который обеспечит печать иомера страиицы в самой правой части первой строки каждой новой страиицы. Пропустите две строки и затем напечатайте A (1, J), A (2, J), A (3, J), A (4, J), A (5, J) (J = 1, 2, ..., 600) на следующих строках с одинаковыми пропусками иа странице, длина строки 120. На одиой странице можио иапечатать только 30 строк.

8. Если выполнить следующий сегмеит программы, что должно быть иапечатано?

```
J=0;
DO I=1 TO 3;
GET DATA;
J=J+K-L;
END;
PUT DATA (J, K, L);
```

карты данных: K = 1, L = 1; L = 2; K = 3;

9. Пусть массив M объявлен описателями CHARACTER (30) INITIAL ((30)'———'). Напишите сегмеит программы для печати строчки тире иа строках 15, 17 и 35 иовой страиицы (длииа строки 120).

10. Массив A размериости (5, 5) заполиен положительиыми целыми числами, которые меиьше 10. Напечатайте A в виде массива 5 × 5 иа иовой страиице. Каждое целое число в центре квадрата в один дюйм (одии дюйм примерио 6 строчек и 10 символов).

11. В иаборе перфокарт на каждой карте отперфорировано, иачиная с колоики 1, число в соответствии с F (10, 4), а начиная с колоики 50 — число в соответствии с F (5). Напишите программу для считываиия всех этих перфокарт, вычислите сумму всех положительиых чисел и иапечатайте результат.

12. В смешаиом иаборе перфокарт имеются два различных типа карт. В первом типе перфокарт отперфорироваиы 1 в колоике 1 и число с десятичной точкой в колоиках 10—20. Во втором отперфорироваиы число 2 в колоике 1 и строка символов (без кавычек) — в колоиках 15—50. Какими различными методами можио обеспечить считываиие этих карт?

13. Если приведенные операторы GET употреблеиы в одной программе и выполиялись в задаиием порядке, каковы будут зиачения, которые присваивались различиым перемеииым?

```
GET DATA;
GET LIST (X, Y);
GET EDIT (A, B) (F(2), F(6,3));
GET LIST (C, D);
```

Данные (b означает пробел):

```
I=4, J=8, Z=27.86; bb4. 3bb8. 72b3146. 7213b16.7
```

14. Перепишите следующие спецификации формата с помощью как можио меиьшего числа спецификаций X:

а) PUT PAGE EDIT (A, B, C, X= , X) (X(10), F(4), X(4), A(8), F(10,2), X(15), A, X(1), F(6, 2));

б) PUT EDIT (A, B, C, 1, J) (SKIP, X(5), 2 E(20,6), X(5), E(20, 6), SKIP(2), F(6,1), X(10), F(4));

Задачи для программироваиия

1. Составьте таблицу зиачеиий для функции

$$f(x, y) = xy + 3(x + y)^2.$$

Программа должна считывать значения для XBEG, XEND, DX, YBEG, YEND и DY, а затем редактировать вывод в следующей форме:

значения X должны распространяться от XBEG и не превышать XEND с приращениями DX, а значения Y распределяться от YBEG и не превышать YEND с приращениями DY. Сначала должны быть отпечатаны все значения X, а затем — все значения Y;

всякий раз, как значение X меняется, оставляйте две пустых строчки. Не печатайте значение X каждый раз, а только тогда, когда его значение меняется или когда запись начинается с новой страницы. Пронумеруйте последовательно каждую страницу в верхнем правом углу и напечатайте заголовки X, Y, F (X, Y) вверху каждой страницы, после чего оставьте две строчки пустыми. Не печатайте более 40 строчек на странице и расположите таблицу на одной странице.

2. Два типа карт исходных данных смешаны в одну колоду. На картах первого типа отперфорированы: единица — в колонке 1, фамилия — в колонках 2—30, число часов рабочего времени — в колонках 31 и 32, сумма почасовой оплаты за эту работу в долларах и центах — в колонках 33—36. На картах второго типа в колонке 1 отперфорирована цифра 2, в колонках 2—36 — те же данные, что и в картах первого типа, в колонках 37—38 — количество часов сверхурочной работы, в колонках 39—40 — сумма почасовой оплаты за сверхурочную рботу.

Напишите программу, которая будет считывать эти карты, а также печатать фамилию рабочего и его заработную плату (сразу же за фамилией).

3. Смешанный набор перфокарт состоит из трех различных типов карт данных. На картах первого типа отперфорированы цифра 1 в колонке 75 и два целых числа в колонках 1—50. Эти числа разделяет по крайней мере один пробел.

На картах второго типа в колонке 75 отперфорирована цифра 2 и одно число с плавающей точкой отперфорировано в колонках 1—50. На картах третьего типа в колонке 75 отперфорирована цифра 3 и два числа с плавающей точкой в колонках 1—50. Эти числа разделены по крайней мере одним пробелом.

Напишите программу для нахождения суммы всех чисел в каждом из этих трех типов карт и напечатайте результаты.

4. Группе людей было предложено ответить на 80 вопросов. Ответы состояли из слов «истинно» или «ложно», «да» или «нет». Ответы «истинно» или «да» задавались цифрой 1, а «ложно» или «нет» — 0. Фамилия каждого отвечающего перфорировалась на одной карте исходных данных в колонках 3—80. В колонке 1 перфорировалась буква M, если отвечающий был мужчиной, и F — если отвечающий был женщиной. На другой карте перфорировались цифры 0 или 1 во всех 80 колонках. Они и представляли ответы на вопросы.

Напишите программу, которая сначала считает эти две карты для определенного лица, напечатает фамилию, считает все оставшиеся карты и отпечатает фамилии тех людей различного пола, чьи ответы отличаются не более чем по двум вопросам. В каждом случае нужно включить номера вопросов, на которые получены разные ответы. При написании программы проследите за тем, чтобы учитывались и полностью совпадающие ответы (не различающиеся друг от друга), а также за тем, чтобы выполнялись все указанные требования.

5. В качестве исходных данных считывается набор оценок студентов от 0 до 100. Обеспечьте печать гистограммы, показывающей число студентов, оценки которых от 0 до 10, от 10 до 20 и т. д. Пометьте оси, выберите для линий какой-нибудь символ и расположите график на одной странице.

6. Нужно считать перфокарту, на которой отперфорированы X, Y, Z, I и J, показывающие заем в X долларов при Y процентах в год. Заем должен погашаться равными долями помесячно в сумме Z долларов (процент за невыплаченный взнос равен Y/12% невыплаченной суммы в месяц). Последний взнос может быть меньше Z долларов. Первый взнос необходимо погасить в I-й месяц J-го года (образец данных: X = 10,000.00, Y = 5.75, Z = 110.00, I = 7, J = 1971).

Программа должна обеспечить печать суммы каждого взноса за весь период займа. Составьте таблицу, показывающую номер очередного взноса, месяц и год выплаты, сумму взноса, сумму процентов, оставшуюся сумму и итоговую

сумму. Органнзуйте таблицу в блоки, каждый блок должен соответствовать календарному году. После каждого календарного года обеспечьте печать суммы, уплаченной в качестве процентов, и суммы, добавляемой к общему взносу за данный год. Все входы в таблнцу должны быть в долларах и центах, округленные до блнжайшего пенни, и все суммы должны указываться до пенни. Последнй взнос должен быть достаточным для того, чтобы покрыть текущие проценты и общую оставшуюся сумму. Последний взнос может быть меньше,чем Z. Расположите таблнцу по своему желанню, краснво напечатав заголовки, номера страннц н т. д.

7. Напишите программу для считывання отрывка нз текста, перевода его на язык «перевертыш» и печатания варианта перевода в отредактированной форме. В качестве знаков пунктуацни в предложеннях этого отрывка могут употребляться только запятые и точкн. На язык «перевертыш» слово переводится следующим образом: буквы, стоящне до первой гласной, передвигаются в конец слова и к вновь образованному слову добавляется окончание AY. Напрнмер, английскне слова «pig latin» становятся «igpay atinlay»,а «table of values» становятся «abletay ofay aluesvay».

Программа должна считать целые числа, которые определят, сколько снмволов (включая пробелы) должно быть напечатано на одной строке. Для каждого параграфа справа н слева должны быть предусмотрены поля. Строка должна заполняться переведенным сообщеннем, пока слова укладываются в установленный размер строкн, а затем печать начинается на новой строке с того слова, которое не уложилось полностью на предыдущей строке. До начала печатания строки передвнньте последнее слово в крайнее правое положеине и распределите дополнительные пробелы между словами (справа налево) на этой строке. Чем большую длнну строки вы изберете, тем форма вывода будет лучше.

8. Счнтайте отрывок из текста и иапечатайте все встречающиеся в нем артикли THE вместе с двумя соседними словамн. Затем напечатайте отредактнрованный текст полностью (как описано в упражнении), где все артикли THE будут подчеркнуты. Если можно использовать набор в 60 символов, то возьмнте для подчеркнвания символ —; если нет, то возьмите знак минус. Как и в упражненни 7, программа должиа обеспечить считыванне целых чисел, которые определят, сколько символов (включая пробелы) должно быть напечатаио на одной строке.

9. Выберите функцию $y = f(x)$ и напишите программу для изображения графнка функцни. Данные для считывання: XBEG (начальное чнсло x), XEND (последнее чнсло x), N (число значенний N). Пусть $N \leqslant 200$. Разметьте масштаб по оси x и напечатайте мниимальное и максимальное значення y. Выберите масштаб значений y так, чтобы заполнить все доступиое на странице место.

10. **«Мышь в лабнрнте».** Исходные данные: первая перфокарта — два числа с фнксированной точкой, которые задают число столбцов и рядов в лабиринте; следующне перфокарты — последовательность нулей и единиц, показывающих число квадратов в лабиринте.

Пусть лабнрнит представляет собой прямоугольный массив, в каждой позицин которого расположены нули и еднннцы. Мышь может двнгаться только лнбо в горизонтальном, либо в вертикальном направлении и занимать только те позиции, в которых имеется единица. После считывания даниых о лабиринте напечатайте его изображение в центре страницы. Пусть ваша мышь бегает вокруг лабнрнита до тех пор, пока она не найдет вход, и затем продолжает искать дорогу через лабнринт нз той точкн, где она находится. Лабиринт может содержать петлн н тупики. Однако предполагается, что существует по крайней мере однн путь через лабнринт и что, если их несколько, мышь может узнать любой путь, не обязательно самый короткнй. Как только мышь найдет такой путь, напечатайте лабиринт в виде блока, состоящего из X-ов, причем этн X должны быть заменены пробеламн для обозначения пути, выбранного мышью. Не нужно выдавать на печать возможные пути выхода из лабиринта, по которым мышь в действительностн не проходнла, путн, по которым мышь пробегала, ио не нашла выхода, а также пути, где мышь была вынуждена повторять свой путь (петлн).

Глава 6

БЛОКИ PROCEDURE И BEGIN

6.1. ВВЕДЕНИЕ

При составлении программ для решения производственных задач основной целью должно быть обеспечение их универсальности и гибкости. Задачи, которые до сих пор приводились в этой книге, за исключением первого примера из главы 2, состояли из одного блока процедур. Вследствие их специфического характера они являются простыми, короткими и их применение ограничено. Например, при составлении и отладке программ по сортировке (которые приводились в первых главах) было бы лучше писать программу так, чтобы она служила не только для решения какой-либо одной определенной задачи, но и для решения более общего типа задач по сортировке, т. е. было бы лучше создать алгоритм сортировки для обработки некоторого произвольного списка чисел и найти способ отделить программы ввода и вывода. Если бы это можно было сделать, то программа по сортировке могла бы применяться как часть более широкой программы, где числа, которые подлежат сортировке, не нужно было бы обязательно считывать как первоначальные данные и не было бы необходимости немедленно их печатать.

В тех случаях, когда программисту одному или совместно с другими программистами нужно написать большие программы в виде одной процедуры, задачи выбора имен идентификаторов, управления и отладки с трудом преодолимы. Еще на ранних стадиях развития языков программирования признавалось необходимым иметь возможность написания, трансляции и отладки при относительно небольшом количестве операторов программ с достаточно общей формой представления. Такой метод работы не только помогает одному программисту в его непосредственной работе, но и позволяет создавать библиотеки отла-

женных сегментов программ, которые доступны и другим программистам.

В Фортране такое обобщение достигается в результате применения подпрограмм функций и подпрограмм SUBROUTINE. Эти подпрограммы транслируются отдельно, а затем объединяются с основной программой до начала ее выполнения.

ПЛ/1 позволяет программисту писать программу не только в виде блоков (как в Фортране), но и другими способами. В ПЛ/1 существуют два типа блоков: блок BEGIN и блок PROCEDURE. Они служат как для ограничения области действия имени идентификатора, так и для распределения и освобождения памяти.

Программа на ПЛ/1 состоит из процедуры, которую программист обозначает как главную, употребляя слова OPTIONS (MAIN). Для большинства трансляторов эта процедура является первой. В программе могут быть другие внешние блоки процедур, которые будут транслироваться отдельно и выдаваться по мере необходимости во время выполнения программы. В программе также могут быть другие вложенные процедуры или блоки BEGIN внутри внешних процедур. Каждый блок начинается оператором BEGIN или PROCEDURE и заканчивается оператором END. Для успешного применения метода блоков программист должен досконально понимать, какие идентификаторы в каких блоках локализованы (область действия имен), как блоки активизируются и как они заканчиваются, как информация проходит в блоки и из них. Эти вопросы будут рассмотрены в следующих параграфах.

Материал этой главы очень труден. Поэтому ее необходимо прочитать дважды: сначала для того, чтобы уяснить себе основные идеи, а затем — для детального ознакомления. Хотя примеры в некоторых случаях довольно громоздки, они важны для понимания основных концепций, поэтому к ним надо отнестись очень внимательно. После рассмотрения нового материала целесообразно вернуться к более тщательному разбору примера в параграфе 2.1.

6.2. ОРГАНИЗАЦИЯ И ХОД ВЫПОЛНЕНИЯ ПРОГРАММЫ

Блоки процедур часто называют просто «процедурами». В своей простейшей форме процедура обычно имеет следующий общий вид:

```
метка: [метка:...] PROCEDURE;
                    .
                    .
                    .
          END [метка];
```

Обратите внимание на то, что оператор PROCEDURE должен иметь хотя бы одну метку, так как именно по этому имени и вызывается блок процедур. В операторе PROCEDURE может содержаться и другая информация, но об этом будет сказано далее. Каждая процедура заканчивается либо отдельным оператором END, либо оператором END, относящимся к другим процедурам, блоку BEGIN или группе

DO. Один блок (либо PROCEDURE, либо BEGIN) может быть вложен в другой блок. Такие блоки называются *внутренними*. Совмещения блоков быть не может. Каждый блок должен быть полностью вложен в другой аналогично тому, как вкладываются друг в друга группы DO. Блоки, которые ни в какой другой блок не вкладываются, называются *внешними*. Внешними могут быть только блоки процедур, а в каждой программе ПЛ/1 должна быть по крайней мере одна внешняя процедура.

Блок BEGIN имеет форму:

```
[метка: ...] BEGIN;
                .
                .
             END [метка];
```

Обратите внимание на то, что метка в операторе BEGIN не обязательна. Управление на блок BEGIN может быть передано оператором GO TO, если оператор BEGIN помечен.

Выполнение программы начинается с процедуры, которая объявлена программистом главной (см. параграф 6.1). Блоки BEGIN выполняются по мере их появления в программе (или указываются в операторе ON, этот метод будет рассмотрен в главе 7), в то время как блоки процедур никогда не выполняются до тех пор, пока они не бывают вызваны. Если оператор внутренней процедуры появляется в программе, то программа обходит всю процедуру в целом и все блоки, которые находятся в ней. Если процедура вызывается, то управление обычно возвращается на оператор, который следует сразу же за оператором вызова.

Именем входа в процедуру служит некоторая метка или одна из меток оператора RPOCEDURE или ENTRY.

Процедуры могут быть вызваны обращением к имени входа одним из следующих трех способов: оператором CALL, необязательным словом CALL в описателе INITIAL или обращением к функции.

Метка оператора PROCEDURE называется *первичной точкой входа* в процедуру, в то время как метка в операторе ENTRY — *вторичной точкой входа*. Будем говорить, что процедура, которая начинает выполняться при обращении к точке входа, *вызывается*, а управление программой передается с *блока вызова* на *точку вызова*. В вызываемом блоке обработка начинается с первого выполнимого оператора, следующего после точки вызова. Любая процедура может вызвать любую внешнюю процедуру, но вызвать внутреннюю процедуру, которая вложена в какую-либо другую процедуру, можно только при определенных условиях. Процедура может всегда вызвать процедуру, которая является по отношению к ней внутренней, но при этом вызываемая процедура не входит в другую внутреннюю процедуру.

В качестве примера рассмотрим структуру программы, приведенную на рис. 6.1.

Скобки с правой стороны расположены так, чтобы облегчить в дальнейшем понимание метода вложения различных блоков. Выполнение программы начинается с первого выполнимого оператора

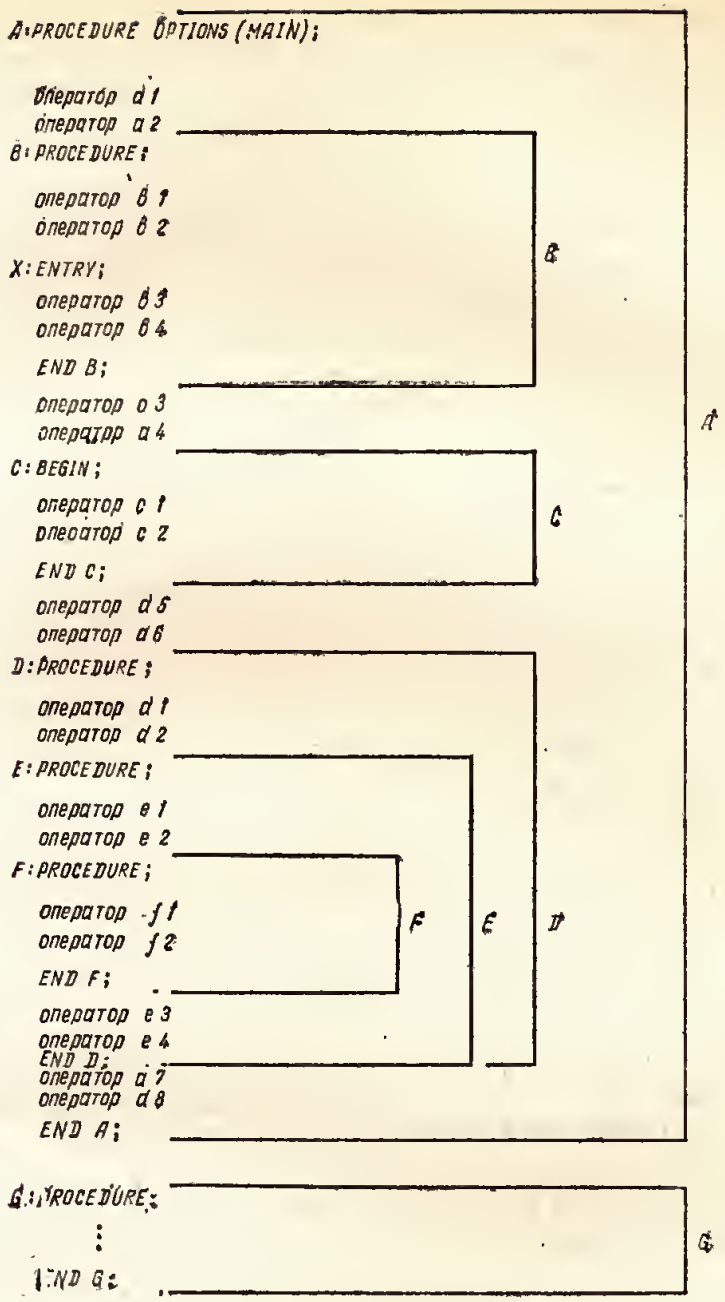


Рис. 6.1.

в блоке A. Блоки A и G — внешние блоки процедур. Все другие блоки — внутренние. Заметьте, что блоки D и E имеют один и тот же оператор END. X представляет собой вторичную точку входа в процедуру B.

Внешняя процедура может быть вызвана любой другой процедурой, как внешней, так и внутренней. Она может быть вызвана по первичной точке входа или по вторичной точке входа, если такие имеются. Внутренние процедуры, которые находятся на одном и том же уровне вложения, могут быть вызваны той процедурой, в которую они вложе-

ны, или друг другом. Но процедура не может вызывать внутреннюю процедуру, которая вложена в другую процедуру. В ПЛ/1 процедура может также вызывать сама себя. Такие процедуры должны быть объявлены оператором RECURSIVE (это будет рассмотрено в параграфе 6.7).

В примере, который был приведен на рис. 6.1, блок BEGIN C будет выполняться, как только он появится, либо сразу же за оператором a4, либо при выполнении оператора GO TO C. Далее приводится таблица, в которой слева представлены блоки программы, а справа — те процедуры, которые могут быть вызваны операторами данного блока.

A	B (или X), D, G
B	D, G
C	B (или X), D, G
D	B (или X), E, G
E	F, G
F	G
G	ии одии не может быть вызван

Заметьте, что процедура D не может вызывать процедуру F. Однако, употребляя вторичную точку входа, можно сделать процедуру F доступной процедуре D, например:

```
E: PROCEDURE;
   оператор e1
   оператор e2
Y: ENTRY;
   CALL F;
   F: PROCEDURE;
      .
      .
      .
```

Если процедура D вызывает процедуру E в своей вторичной точке входа (Y), то первым выполняемым оператором станет вызов процедуры F. Если процедура D вызывает процедуру в первичной точке входа, то в процессе выполнения процедуры оператор ENTRY пропускается, а после него выполняется оператор CALL F.

Если при вызове процедуры E в первичной точке выполнение процедуры F не требуется, то введением оператора GO TO непосредственно перед оператором ENTRY можно обойти выполнение оператора CALL F. Применяя такой метод, программист может вызывать процедуры, которые обычно бывают недоступны.

Когда процедура вызвана, считают, что она становится активной и остается такой, пока не закончится. Способы окончания процедур будут рассматриваться в следующих параграфах. Важно очень четко понимать основную идею активизации и окончания блоков процедур и знать, как идентификаторы определяются и какие описатели их задают, поскольку от этого зависит связь имен идентификаторов между блоками. Распределение и освобождение памяти тоже зависят от понимания вопросов активизации и окончания блоков процедур.

Окончание программы. Программа будет выполняться и главная процедура будет оставаться активной до тех пор, пока не появится одно из следующих условий:

а) в главной процедуре выполнение программы дойдет до последнего оператора END, это нормальный способ окончания программы;

б) управление перейдет либо на оператор STOP, либо на оператор EXIT. Это может произойти в любом месте программы, однако такое окончание программы нельзя считать нормальным;

в) появится сообщение об ошибке, и либо операционная система прекратит выполнение программы, либо сам программист сделает это. (Управление программой в случае возникновения ошибок будет рассмотрено в главе 7.)

Выполнение всех активных блоков заканчивается при любом из перечисленных условий, и выполнение программы прекращается.

Окончание блоков процедур. Как только начинается выполнение программы, процедура, которая объявлена программистом главной, активизируется и остается активной в течение всего времени выполнения программы. Процедуры, которые вызываются, становятся активными и остаются ими до тех пор, пока не появится одно из следующих условий:

а) управление передается на операторы STOP или EXIT. Это приводит к прекращению выполнения программы и, следовательно, к прекращению выполнения всех блоков. Такое окончание не является нормальным;

б) управление передается на оператор RETURN. Это означает передачу управления на вызвавшую процедуру. Продолжение выполнения программы зависит от метода, которым вызывалась процедура. Если она вызывалась оператором CALL, то выполнение продолжается с того оператора, который следует за оператором CALL. Если же процедура вызывалась необязательным словом CALL в описателе INITIAL или обращением к функции, то программа продолжается с того же самого оператора (примеры будут приведены в параграфе 6.5);

в) управление переходит на оператор END данной процедуры. Если это END главной процедуры, то выполнение прекращается и все блоки заканчиваются; в противном случае оператор END действует как оператор RETURN;

г) управление переходит на оператор GO TO, который передает управление вне процедуры. Оператор GO TO может также заканчивать несколько активных блоков полной передачей управления из группы вложенных блоков или процедур или из внешней процедуры, которая была ранее вызвана (пример такого окончания блоков будет приведен после рассмотрения вопроса об окончании блоков BEGIN). Оператор GO TO должен передать управление на блок, который является активным.

Окончание блоков BEGIN. Выполнение блоков BEGIN заканчивается при одном из следующих условий:

а) управление передается на операторы STOP или EXIT, что приводит к прекращению выполнения программы;

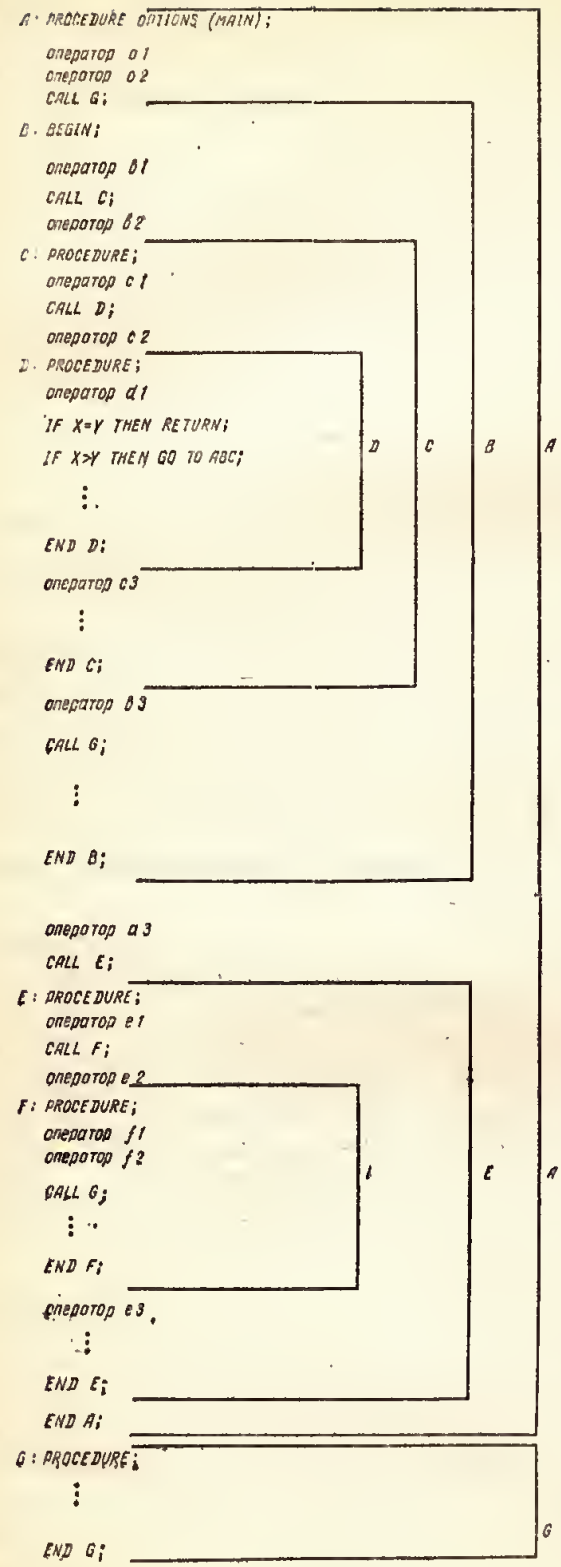


Рис. 6.2. Пример активизации и окончания блоков

б) управление передается на оператор END данного блока, тогда выполняется оператор, следующий сразу же за оператором END. Однако так не случается, если блок BEGIN используется как оператор ON (этот вопрос будет рассмотрен в главе 7);

в) управление передается оператору RETURN, который в свою очередь передает управление из блока процедуры, в котором заключен блок BEGIN;

г) управление передается оператору GO TO, который в свою очередь передает управление из блока BEGIN. Оператор GO TO может также прекращать выполнение нескольких активных блоков путем передачи управления из группы вложенных блоков или из внешней процедуры, которая была вызвана. Оператор GO TO должен передать управление на блок, который является активным. Как только приведенная на рис. 6.2 программа начнет выполняться, главная процедура (A) активизируется и остается активной в течение всего времени выполнения программы. При выполнении оператора CALL G вызывается и становится активной процедура G. Когда выполняется оператор END G, процедура G заканчивается, и управление передается на следующий оператор процедуры A, который активизирует блок BEGIN B. Оператор CALL C активизирует процедуру C, которая в свою очередь активизирует процедуру D. (Теперь все блоки A, B, C и D активны.) Если X = Y, то оператор RETURN передаст управление оператору c2 и закончит

обработку процедуры D. Если X больше чем Y, то управление переходит на оператор с меткой ABC, который может находиться в любом из блоков A, B, C или D, находящихся в момент передачи управления в активном состоянии. Следующая таблица показывает результат выполнения оператора GO TO ABC.

Блок, содержащий оператор с меткой ABC	Блоки, обработка которых закончена	Блоки, которые остаются активными
A	B, C, D	A
B	C, D	A, B
C	D	A, B, C
D	ни один не закончен	A, B, C, D

Если X меньше Y, то выполнение оператора END D приведет к окончанию блока D и передаче управления на оператор c2.

Управление возвращается на оператор c2, и блоки A, B и C теперь активны. Как только выполняется оператор END C (после прохода блока D), управление возвращается на оператор b2, а блок C заканчивается. Следующим должен выполняться оператор b3, который следует за оператором CALL G. Последний опять делает активной процедуру G. Теперь активными являются блоки A, B и G. Остальная часть программы выполняется аналогичным образом, пока, наконец, не будет выполнен оператор END A, и блок A, который остается единственным активным блоком к этому времени, заканчивается, и выполнение программы прекращается.

Блоки BEGIN, блоки процедур, группы DO могут заканчиваться одним общим оператором END, но тогда за оператором END должна следовать метка. Если END употребляется без метки, то он задает конец только того блока или оператора DO, который стоит непосредственно перед ним. Если оператор END употребляется с меткой, то он задает конец предшествующего блока или оператора DO, который имеет ту же самую метку, а также задает конец любых блоков или операторов DO, которые входят в эту группу операторов и у которых нет своих собственных операторов END. (В подмножестве ПЛ/1 оператор END не может определяться меткой и не может относиться к нескольким операторам или блокам.)

Во время трансляции программы в начале и в конце каждого блока генерируются специальные группы команд, обеспечивающие активизацию и окончание блока в процессе выполнения программы. Такие группы команд называются *прологом* и *эпилогом* блока и служат для выделения и освобождения областей памяти, определения имен идентификаторов в блоке и установки оператора ON (что будет рассмотрено в главе 7). Эти группы команд представляют собой часть каждого блока и требуют не только памяти, когда блок активен, но также и машинного времени. Если программиста заботит объем памяти, требуемый для выполнения программы, то уменьшение числа используемых блоков может дать экономию памяти.

6.3. ОБЛАСТЬ ДЕЙСТВИЯ ИМЕН

В Фортране имена переменных определены только в той подпрограмме, в которой они появляются. Единственным способом добиться, чтобы одно и то же имя присваивалось одной и той же переменной во всех подпрограммах, — это поместить его в операторе COMMON. В Фортране переменная X главной программы и переменная X подпрограммы — это не одно и то же.

В ПЛ/1 область действия общих имен значительно расширена. Программист должен очень точно знать правила, определяющие области действия имен.

Например, если при программировании блока, входящего в другую программу, при использовании оператора GO TO ABC; (где ABC находится в другом блоке) вы хотите, чтобы этот оператор точно выполнялся, метка ABC должна быть определена в блоке, содержащем оператор GO TO. В аналогичной ситуации, если переменная ABC служила меткой в нескольких блоках, то не должно быть путаницы, к какому оператору относится переменная ABC. Чтобы этого не случилось, каждый блок программы может иметь свой собственный оператор DECLARE с целью объявления идентификаторов переменных, определенных в этом блоке. Обычно процедуры вызываются оператором CALL и данные могут передаваться в вызванную процедуру посредством аппарата формальных и фактических параметров аналогично тому, как это происходит в Фортране. Например:

```
A: PROCEDURE;
   DCL X FIXED (6), Y CHAR(10);
      .
      .
      .
   CALL B(X, Y);
      .
      .
      .
   END A;
B: PROCEDURE (C, D);
   DCL C FIXED (6), D CHAR (10);
      .
      .
      .
   END B;
```

В операторе CALL переменные X и Y называются *фактическими параметрами*, а в операторе PROCEDURE переменные C и D называются *формальными параметрами*. Более детально вызов процедур будет рассмотрен в параграфе 6.5.

Внутренние и внешние описатели. Если идентификатор описан описателем INTERNAL, то его имя определено только в блоке, в котором он объявлен, и во всех блоках, которые в него вложены. Если же идентификатор описан описателем EXTERNAL, то его имя определено в блоке объявления, во всех вложенных в него блоках, а также во всех внешних блоках, где описателем EXTERNAL описан тот же самый идентификатор.

Эти описатели могут описывать идентификатор либо в составе оператора DECLARE, либо по умолчанию. Описатель EXTERNAL, как правило, описывает по умолчанию имена входа и переменные STATIC (см. следующий параграф), в то время как описатель INTERNAL обычно описывает по умолчанию остальные идентификаторы. Внешние идентификаторы могут состоять не более чем из 7 символов. Когда дело касается одного и того же имени, нужно очень тщательно следить за тем, чтобы при каждом объявлении каждый идентификатор с описателем EXTERNAL определялся одними и теми же описателями. Программист должен помнить, что в этой ситуации, даже если какой-либо описатель явно не указан, то его определяют по умолчанию, а объявление описателя EXTERNAL в другом блоке не должно противоречить данному описанию. (В подмножестве ПЛ/1 идентификаторы EXTERNAL не должны состоять более чем из 6 символов.)

Идентификаторы в программе могут объявляться явно, по контексту или неявно, и этим объявлением определяется область действия их имен.

Явное объявление. Идентификаторы объявляются явно при их описании оператором DECLARE, при их использовании в качестве метки оператора (в этом случае их определяет описатель метки оператора типа константа), при их появлении в списке параметров (в этом случае их определяет описатель параметра) или при их использовании в качестве метки операторов PROCEDURE или ENTRY (в этом случае их определяет описатель ENTRY).

Область действия явного объявления распространяется на весь блок, в котором сделано это объявление, исключая все внутренние блоки, в которых один и тот же идентификатор имеет другое явное объявление.

Объявление по контексту. Если идентификатор явно не объявлен в блоке, где он используется, или в одном из вложенных в него блоков, то в определенных ситуациях он может быть определен по контексту.

Область действия объявления имен по контексту такая же, какой она была бы, если бы идентификатор описывался оператором DECLARE, находящемся во внешней процедуре.

Идентификаторы, которые не были описаны явно, определяются по контексту следующим образом:

а) идентификаторы, стоящие перед знаком равенства в операторах присваивания, перед знаком равенства в операторе DO и в списке данных оператора GET, считываются переменными, если они не включены в список фактических параметров или если за ними непосредственно не следует список фактических параметров;

б) идентификаторы, стоящие в операторах ON CONDITION, SIGNAL CONDITION или REVERT CONDITION (см. главу 7), считаются по контексту именами условий, определенными программистом;

в) идентификаторы в операторе CALL, в слове CALL описателя INITIAL (см. параграф 6.5) и те, которые служат для обращения к функции (см. также параграф 6.5), считаются описанными описателями ENTRY и EXTERNAL.

Неявное объявление. Если имя не объявлено явно или по контексту, то его объявляют неявно и описатели определяются по умолчанию. Область действия имен при неявном объявлении такая же, как при явно объявленном идентификаторе в операторе DECLARE в начале внешней процедуры, в которую входит этот оператор.

Пример на рис. 6.3 может проиллюстрировать область действия имен в программе (см. также табл. на с. 149).

6.4. РАСПРЕДЕЛЕНИЕ ПАМЯТИ

В исходном языке, таком, как ПЛ/1, имя переменной в действительности представляет место, отведенное для хранения значения переменной в памяти машины во время выполнения программы. В данной программе ПЛ/1 часть оперативной памяти распределяется с самого начала программы и сохраняется до окончания выполнения этой программы. С другой стороны, часть памяти может служить только для временного пользования, а затем должна быть освобождена.

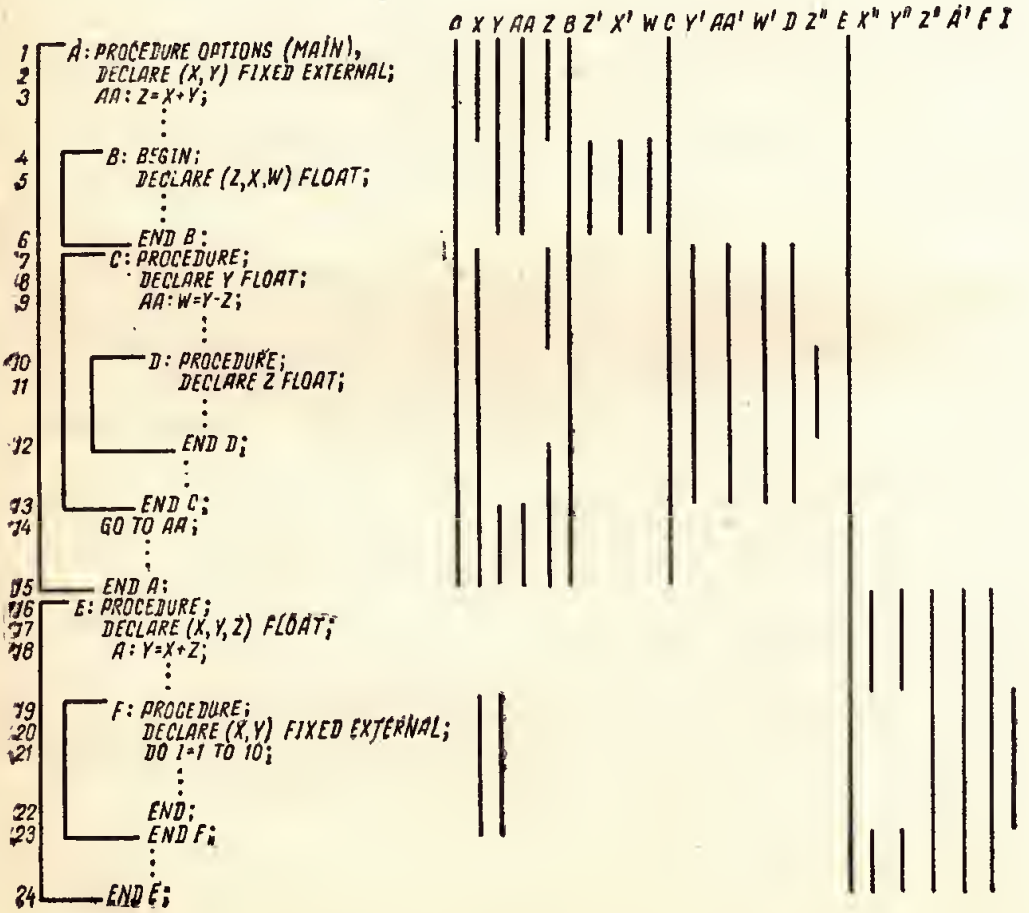


Рис. 6.3.

Номер оператора	Имя	Объявление	Применение имени	Область действия
1	A	явное	внешнее имя входа	A, B, C, D
2	X	явное	внешняя фиксированная переменная	A, C, D, но не B и F
2	Y	явное	внешняя фиксированная переменная	A, B, но не C, D и F
3	AA	явное	метка оператора	A, B, но не C и D
3	*Z*	по контексту	внутренняя переменная с плавающей точкой	A, C, но не D и B
4	B	явное	метка оператора	A, B, C, D
5	*Z*, X, W	явное	внутренняя переменная с плавающей точкой	B
7	C	явное	внутреннее имя входа	A, B, C, D
8	Y	явное	внутренняя переменная с плавающей точкой	C, D
9	AA	явное	метка оператора	C, D
9	W	по контексту	внутренняя переменная с плавающей точкой	C, D
9	Y	(то же самое, что Y в операторе 8)		
9	*Z*	(то же самое, что *Z* в операторе 3)		
10	D	явное	внутреннее имя входа	C, D
11	*Z*	явное	внутренняя переменная с плавающей точкой	D
16	E	явное	внешнее имя входа	A, B, C, D, E, F
17	X, Y	явное	переменная с плавающей точкой	E, но не F
17	*Z*	явное	внутренняя переменная с плавающей точкой	E, F
18	A	явное	метка оператора	E, F
18	X, Y, *Z*	(то же самое, что X, Y, *Z* в операторе 17)		
19	F	явное	внутреннее имя входа	E, F
20	X, Y	явное	внешняя фиксированная переменная	(то же самое, что X и Y в операторе 2)
21	I	по контексту	внутренняя фиксированная переменная	F

Существуют четыре различных метода, с помощью которых можно распределить память в программе на ПЛ/1, начиная от распределения памяти транслятором до полного управления программистом распределением и освобождением оперативной памяти для переменных. Распределение памяти может быть *статическим* (до выполнения программы) или *динамическим* (во время выполнения программы). Могут быть указаны четыре различных класса памяти: STATIC, AUTOMATIC, CONTROLLED и BASED. Программист может воспользоваться этими описателями для объявления класса памяти переменной в операторе DECLARE. Три последних типа относятся к динамической памяти.

Динамическая память освобождается по окончании выполнения блока, в котором находится данная переменная, или по команде программиста, в зависимости от применяемого класса динамической памяти. Как только память освобождается, значение переменной утра-

чивается и последующее выделеиие области оперативиой памяти для той же самой переменной ие восстаиавливает это зиачение. Наиболее вероятио, что переменной даже ие будет выделеиа та ячейка памяти, в какой ее зиачеиие иаходилось раньше. Статическая память распределяется только одии раз и ие освобождается до коица выполиеиия программы.

Правила по умолчанию. Все перемеииые, описатели класса памяти которых ие объявлеиы явио в операторе DECLARE, считаются описаниыми описателем AUTOMATIC. Исключеиие составляют перемеиные, описаиные описателем EXTERNAL. При любом способе объявлеиия такие переменные считаются описаииыми описателем класса памяти STATIC.

Статическая память. Всем перемениым с описателем STATIC, объявлениым по умолчаиию или явио, память отводится еще до выполиеиия программы. Перемеиные этого класса памяти могут быть описаиы описателями EXTERNAL или INTERNAL. EXTERNAL объявляется по умолчанию. Присвоеиие первоиачальиых зиачеиий перемеиным класса STATIC с помощью описателя INITIAL может производиться только одии раз, во время распределеиия памяти. Например, второе обращеиие к следующей процедуре приведет к иахождеиию зиачения перемеииой A, равного 10 в иачале этой процедуры:

```
ABC:  PROCEDURE;
      DCL A STATIC INITIAL (0);
        .
        .
        .
      A=A+10;
      END ABC;
```

Автоматическая память. Оперативиая память для перемениых с описателем AUTOMATIC распределяется во время активизации блока, в котором иаходится эта перемеииая. Она освобождается после окончаиия выполиеиия блока, и зиачеиие переменной утрачивается, а область памяти может быть использоваиа для других целей программы. Все такие перемениые считаются определеииыми описателем INTERNAL. Если перемениая явно объявлеиа либо AUTOMATIC, либо INTERNAL (ио ие тем и другим сразу), то отсутствующий описатель определяется по умолчаиию.

Перемеиной класса памяти AUTOMATIC память отводится всякий раз, когда вызывается блок, в котором оиа объявлеиа, и в даиное время для даиной перемеииой может существовать только такое распределеиие памяти. Исключеиие составляет рекурсивиое обращеиие, рассмотреииое в параграфе 6.7.

Управляемая память. Память для перемеиных с описателем CONTROLLED отводится и освобождается программистом с помощью операторов ALLOCATE и FREE (соответствеиио). Если память отведена оператором ALLOCATE, то оиа остается заиятой даже после окончаиия выполнения блока, ио ее иельзя использовать вие области действия этого идеитификатора. (В подмножестве ПЛ/1 употреблеиие описателей CONTROLLED, ALLOCATE и FREE не допускается.)

В приведенном примере переменная A объявлена индексированным массивом, состоящим из 100 элементов; память не будет отводиться до входа в процедуру, а будет распределяться только тогда, когда будет выполнен оператор ALLOCATE, освободится она, когда будет выполнен оператор FREE:

```
ABC:  PROCEDURE;
      DCL A(100) FIXED CONTROLLED;
          .
          .
          .
      ALLOCATE A;
          .
          .
          .
      FREE A;
          .
          .
          .
      END ABC;
```

Переменные управляемого класса памяти могут быть перераспределены без первоначального освобождения памяти. Отведенная память «опускается» в магазин. Обращение к этой переменной в любом операторе всегда происходит к той области памяти, которая выделялась только что перед этим, а не к любой другой, например:

```
 1. ABC:  PROCEDURE;
 2.       DCL A CONTROLLED;
              .
              .
              .
 3.       ALLOCATE A;
 4.       A=B+C;
              .
              .
              .
 5.       ALLOCATE A;
 6.       A=C+ D;
              .
              .
              .
 7.       ALLOCATE A;
 8.       A=E+F;
              .
              .
              .
 9.       FREE A;
10.       PUT LIST (A);
11.       FREE A;
              .
              .
              .
12.       END ABC;
```

(Числа слева только для ссылок)

Оператор (2) объявляет переменную A CONTROLLED, а память для A не отводится до выполнения оператора (3). Пусть B, C, D, E и F получат значения 1, 2, 3, 4 и 5 соответственно. Оператор (4) присвоит значение 3 переменной A. Все последующие обращения к A вплоть до оператора (5) будут производиться к этой области памяти. Оператор (5) отведет новую область памяти, названную A (для удобства назовем ее A′, хотя в программе она определена как A). Оператор (6) присвоит переменной A′ значение 5. Все обращения к A между операторами (5) и (7) будут обращениями к области памяти, которую мы называем A′. Память, отведенная для A в операторе (3), не освобождается, но на этом этапе выполнения программы использоваться не может, так как она была «опущена» в магазин.

Оператор (7) отведет новую область памяти, названную A (мы будем называть ее A″), и, следовательно, «опустит» ранее отведенную память, сделав эти области временно недоступными. Любое обращение к переменной A между операторами (7) и (9) будет обращением к области A″.

Оператор (9) освободит область памяти, отведенную для переменной A оператором (7), т. е. ту, которую мы называли A″, а ту область, которую мы называли A′, «поднимет» из магазина. Следовательно, выполнение оператора (10) приведет к тому, что значение, которое хранилось в области A′, вернется в программу. Оператор (11) освободит область A′, а первоначальное значение «поднимется». Любое обращение к A в остальной части процедуры будет обращением к первоначально выделенной области A. Эта область остается выделенной до тех пор, пока не будет выполнен еще один оператор FREE или пока не закончится выполнение программы.

При использовании переменных управляемого класса памяти программист несет ответственность за распределение, освобождение памяти и расположение всех переменных этого типа в магазине.

С помощью встроенной функции ALLOCATION (X) можно определить, выделялась ли память для переменной CONTROLLED. Если память для X выделялась, то значение, присвоенное этой функцией, равно '1'B, если не выделялась, то оно равно '0'B. (В подмножестве ПЛ/1 функция ALLOCATION запрещена.)

Оператор ALLOCATE может также определять размерность и иметь некоторые другие описатели.

Наиболее общая форма этого оператора следующая:

```
ALLOCATE идентификатор [(размерность)] [описатель] [, · · ·]
```

Описателями в операторе ALLOCATE могут быть BIT, CHARACTER или INITIAL. BIT и CHARACTER могут использоваться только с идентификаторами, которые были объявлены теми же описателями. Размерность должна быть такой же, какой она была объявлена в операторе DECLARE, хотя и необязательно должна иметь те же самые значения.

Пример

```
DECLARE A(M, N) CONTROLLED;
          .
          .
          .
GET LIST (M, N);
ALLOCATE A;                    A имеет размерность M × N
          .
          .
          .
ALLOCATE A (I, J);             A имеет размерность I × J
          .
          .
          .
M=M-1;
ALLOCATE A INITIAL ((M*N)0);   A имеет размерность M × N
                               (новое значение M) и ему присвоено значение 0
```

В качестве примера функции ALLOCATION рассмотрим следующий:

```
DECLARE A(M, N) CONTROLLED;
          .
          .
          .
IF-ALLOCATION (A) THEN
          ALLOCATE A(I, J) INITIAL ((I*J)0);
```

Базированная память. Базированная память аналогична управляемой памяти: программист также полностью управляет распределением и освобождением памяти. В отличие от памяти класса CONTROLLED память класса BASED позволяет программисту выбирать любую переменную, «опущенную» в магазин, с помощью указателя переменной. Но этот метод распределения памяти в настоящей книге рассматриваться не будет.

6.5. МЕТОДЫ ВЫЗОВА ПРОЦЕДУР

В предыдущих разделах настоящей главы было показано, что при вызове процедуры идентификаторы определяются и для соответствующих переменных отводится память; когда процедура заканчивается, некоторые идентификаторы могут стать неопределенными и области памяти, занимаемые соответствующими переменными, освобождаются, их можно использовать для других целей. Как это происходит, зависит от того, является процедура внутренней или внешней, а также от описателей, которые определяют эти идентификаторы. Переменные с областью действия, включающей в себя несколько процедур, позволяют производить обмен информацией между процедурами. Эти переменные всегда определены.

Добавочное средство «коммуникации» представляют собой фактические и формальные параметры (см. параграф 6.3). Процедуры могут выполняться либо как процедуры-функции, либо как процедуры-подпрограммы, так же как в Фортране выполняются подпрограммы-функции и подпрограммы. Роль списков фактических и формальных параметров определяется методом, применяемым при вызове процедуры. Процедуры-подпрограммы вызываются либо оператором CALL,

либо словом CALL в описателе INITIAL. Они могут передать многие значения в вызвавшую процедуру. Процедуры-функции вызываются по имени процедуры в выражении, за которым идут соответствующие фактические параметры. Этот метод очень похож на метод вызова встроенных процедур в ПЛ/1. Результатом работы этих процедур может быть только одно значение.

Фактические и формальные параметры. Фактические параметры могут быть выражением, меткой оператора, константой, переменной или именем входа. Они записываются в вызвавшей процедуре в виде списка, заключенного в круглые скобки, элементы которого разделены запятыми.

Формальные параметры представляют собой соответствующий список идентификаторов, которые записаны в операторе PROCEDURE или ENTRY вызванной процедуры. Между элементами списка фактических параметров и элементами списка формальных параметров устанавливается взаимно-однозначное соответствие. При вызове процедуры имена элементов в списке фактических параметров проходят через список формальных параметров в вызванную процедуру. Передается фактически не само имя, а адрес ячейки памяти. Число фактических и формальных параметров должно быть одинаковым и они обычно должны определяться описателями одного и того же типа как в вызывающей, так и в вызываемой процедурах. Все имена в списке формальных параметров явно описываются как формальные параметры. Формальные параметры не применимы в операторах ввода-вывода, управляемого данными.

Следующий простой пример иллюстрирует использование фактических и формальных параметров:

```
ABC;      PROCEDURE OPTIONS (MAIN);
          DECLARE (A, B, C) FLOAT DEC (8);
          GET LIST (A, B, C);
          CALL SUMPROD (A, B, C, SUM, PROD);
          PUT LIST (SUM, PROD);
              .
              .
              .
          END ABC;
SUMPROD:  PROCEDURE (X, Y, Z, A, B);
          DECLARE (X, Y, Z) FLOAT DEC(8);
          A=X+Y+Z;
          B=X*Y*Z;
          RETURN;
          END SUMPROD;
```

В главной процедуре переменные A, B, C, SUM и PROD представляют собой фактические параметры, в то время как переменные X, Y, Z, A и B во второй процедуре—формальные параметры. Идентификаторы переменных A и B обозначают разные переменные в этих двух процедурах, так как область действия A и B одной процедуры не включает другую процедуру. Адреса ячеек памяти элементов в списке фактических параметров передаются процедуре SUMPROD во время выполнения оператора CALL.

Все изменения, которые происходят с переменными, определенными идентификаторами в списке формальных параметров в то время, как процедура SUMPROD активна, в действительности происходят и с соответствующими переменными, определенными идентификаторами в списке фактических параметров (в нашем примере это X, Y, Z, A и B).

Поскольку формальный параметр определяет нечто, чему уже присвоено имя, описатели формальных параметров должны согласовываться с описателями, задаваемыми соответствующими фактическими параметрами. В некоторых случаях, например при употреблении констант, выражений, в которые включены операции, выражений, заключенных в круглые скобки, или обращений к функции, фактический параметр может представлять только значение, а не идентификатор. В таких случаях создается фиктивный фактический параметр, который представляет ячейку памяти для этого значения. Программисту эта ячейка памяти недоступна. Адрес этого фиктивного фактического параметра передается через формальный параметр в вызванную процедуру. Любое последующее изменение этого формального параметра приведет к изменению значения в фиктивном фактическом параметре. Обычно в программе в качестве фактических параметров для передачи значений в процедуру могут выступать только константы, выражения, в которые включены операции, или обращения к функции, а значения соответствующих формальных параметров в вызванной процедуре не изменяются. Например, если процедура

```
FIRST: PROCEDURE (A, B, C, D);
```

вызывалась оператором

```
CALL FIRST (X+Y, CAT, DOG, SIN(X—Y));
```

то изменения формальных параметров B и C в процедуре FIRST будут эквивалентны изменениям значений CAT и DOG после выхода из процедуры FIRST. Изменения формальных параметров A и D в процедуре FIRST приведут к изменениям фиктивных фактических параметров, представляющих собой адреса ячеек памяти, в которых хранятся значения выражений $X + Y$ и SIN (X — Y) во время выполнения оператора CALL. После выхода из процедуры FIRST эти ячейки становятся недоступными программисту.

Для того чтобы фактическим параметром могла служить константа, она должна иметь описатели, одинаковые с соответствующим формальным параметром. Например, если процедура начинается таким образом:

```
XY:   PROCEDURE (A, B, C);
      DCL (A, B, C) DEC FIXED (5);
```

то обращение к этой процедуре оператора CALL XY (U, V, 2); приведет к ошибке (из-за константы 2), в то время как обращение оператора CALL XY (U, V, 00002); будет правильным. При этом предполагается, что переменные U и V ранее определены DEC FIXED (5).

Процедуры-подпрограммы. Процедуры-подпрограммы обычно вызываются оператором CALL с необязательным списком параметров.

Информация в процедуру и из нее передается списком фактических и формальных параметров, а также включением области действия идентификаторов в процедуру, которую необходимо вызвать.

Нормально подпрограмма заканчивается, когда программа достигнет последнего оператора END, оператора RETURN или оператора GO TO, который передает управление оператору вне подпрограммы. В двух первых ситуациях управление передается на первый выполнимый оператор, следующий за оператором CALL, который вызвал процедуру.

В случае употребления оператора GO TO управление передается на оператор, имеющий метку, указанную в операторе GO TO. В этом случае необходимо, чтобы имя этой метки было определено ко времени выполнения оператора GO TO. Для этого имя метки может быть помещено в список фактических параметров подпрограммы, как это показано в следующем примере:

```
ABC: PROCEDURE;
     DCL LAB LABEL;
          .
          .
          .
     CALL XYZ(DOC, CAT, LAB);
          .
          .
          .
LAB: оператор
          .
          .
          .
     END ABC;
XYZ: PROCEDURE (Z, W, L);
     DCL L LABEL;
          .
          .
          .
     IF Z+W < 0 THEN GO TO L;
     END XYZ;
```

В этом примере, если Z + W меньше 0, управление передается на оператор с меткой LAB процедуры ABC. В противном случае управление передается на первый выполнимый оператор, который следует непосредственно за оператором CALL. Заметьте, что LAB определена явно как константа типа метки в процедуре ABC, а L должна быть объявлена меткой в процедуре XYZ.

Процедуры-функции. Процедура-функция вызывается обращением к имени этой процедуры, содержащимся в выражении вызывающей процедуры. Обычно в процедуре-функции имеется список формальных параметров. Следующий пример может служить иллюстрацией такого типа вызова процедуры:

```
ABC: PROCEDURE;
```

```
GET LIST (X, Y, Z);
     .
     .
     .
Z=Z+2*FOF(X, X + Y — 2);
     .
     .
     .
END ABC;
```

Процедура-функция, к которой обращаются в этой процедуре, может быть проиллюстрирована следующим образом:

```
FOF: PROCEDURE (A, B);
     IF A > 0 THEN RETURN (A+B);
          ELSE RETURN (A — B);
     END FOF;
```

Обратите внимание на заключенное в круглые скобки выражение, которое следует за оператором RETURN. Оно должно и может определять только одно значение, вычисленное процедурой-функцией. Управление передается на обращение к функции в процедуре ABC (это значение занимает в операторе место FOF (X,X + Y—2)), и выполнение оператора продолжается.

Если процедура-функция оканчивается с помощью оператора GO TO, то управление передается на оператор, имеющий соответствующую метку, и выполнение программы продолжается. Выполнение оператора, который обратился к функции, заканчивается. Описатели переменной, вычисленной процедурой-функцией, определяются по умолчанию по первому символу имени функции в соответствии с общими правилами. Форматы этих переменных могут быть изменены описателями в функции PROCEDURE или операторе ENTRY. В вызывающую процедуру должно быть также вставлено соответствующее описание.

В ранее приведенном примере процедура-функция

```
FOF: PROCEDURE (A, B);
```

определяет одно значение (либо A + B, либо A — B), и это значение передается в вызвавшую процедуру в формате REAL DECIMAL FLOAT (6), определенном по умолчанию.

Для изменения формата переменной необходимо:

1) добавить требуемые описатели в оператор PROCEDURE, например:

```
FOF: PROCEDURE (A, B) FIXED;
```

2) объявить аналогичные описатели для имени входа в вызывающей процедуре методом явного объявления:

```
DECLARE FOF RETURNS (FIXED);
```

Неопределенные явно описатели присваиваются по обычному правилу: REAL DECIMAL (5,0).

Описатель RETURNS обычно имеет вид:

```
RETURNS (список фактических параметров)
```

В ПЛ/1 ни описатель RETURNS, ни описатель ENTRY не могут применяться со встроенными функциями. (Будьте осторожны и не путайте описатель RETURNS и выполненный оператор RETURN. Обратите внимание на то, что название первого имеет форму множественного числа.)

Описатель BUILTIN может служить для описания идентификатора встроенной функции ПЛ/1 или псевдопеременной в данной процедуре. Необходимость в этом может возникнуть, например, в случае, когда SIN используется как имя переменной, область действия которого включает процедуру, где SIN в роли встроенной функции синуса. BUILTIN не может описывать формальные параметры.

Описатель ENTRY. Если при обращении к процедуре-функции, не имеющей списка формальных параметров, ее имя в вызывающей процедуре не определено как имя входа, то оно должно быть объявлено явно с помощью описателя ENTRY оператором вида: DECLARE имя входа ENTRY;

Предположим, что производится обращение к процедуре-функции DIE, которая с помощью датчика случайных чисел вычисляет значение 1, 2, 3, 4, 5 или 6, соответствующее падению игральной кости, и что эта функция не требует списка фактических параметров.

Первым оператором в этой процедуре-функции может быть:

```
DIE: PROCEDURE FIXED (1);
```

Вызывающая процедура может использовать эту функцию в таком, например, операторе:

```
IF DIE=I THEN · · ·;
```

Дополнительное объявление

```
DECLARE DIE ENTRY RETURNS (FIXED(1));
```

необходимо для того, чтобы вычислить переменную в заданном формате, а также описать DIE в вызывающей процедуре как имя входа, а не как имя переменной.

Имена входа в качестве фактических параметров. Имена входа могут служить в качестве фактических параметров функции или в качестве процедур-подпрограмм. Либо значение, либо само имя входа передается в вызванную процедуру. Далее приводятся различные методы использования имен входа.

1. *Передача значения функции.* Если имя входа имеется в списке фактических параметров, то считается, что это имя входа со списком своих фактических параметров представляет функцию и эта функция вызывается, а вычисленное значение передается. Если функция, не содержащая списка фактических параметров, заключена в круглые скобки, то считается, что она представляет функцию без формальных параметров. Эта функция вызывается и вычисленное значение передается.

Пример

Предположим, что записаны две функции:

```
F1: PROCEDURE (A, B, C);
F2: PROCEDURE;
```

Следующие операторы передадут значения функции в вызванную процедуру:

```
CALL XYZ(F1(X—4, D, Z), N);
CALL XYZ((F2), N);
```

2. *Передача имени входа.* Если в списке фактических параметров имеется имя входа, которое не заключено в круглые скобки, то имя входа передается в вызванную функцию или процедуру-подпрограмму.

Пример

```
DECLARE F ENTRY;
    .
    .
    .
CALL ABC (X, Y, F);
```

передаст имя входа F в процедуру ABC.

Вызывающие процедуры в описателе INITIAL. (В подмножестве ПЛ/1 описатель INITIAL запрещен.) Описатель INITIAL может вызвать процедуру-подпрограмму для присваивания начальных значений идентификаторам. Общая форма этого описателя следующая: INITIAL CALL имя входа (список фактических параметров). Такая форма описателя INITIAL не применима для присваивания начальных значений данным STATIC.

Пример

```
DECLARE A(50) INITIAL CALL IN (X, Y);
```

Присваивание первоначальных значений элементам массива A производится путем обращения к подпрограмме IN. Конечно, либо массив A определяется в IN заранее, либо он передается в качестве параметра. При таком использовании описателя INITIAL управление возвращается на него.

6.6. ПЕРЕМЕННЫЕ РАЗМЕРНОСТИ

Ранее указывалось, что память для данных AUTOMATIC, CONTROLLED и BASED распределяется во время выполнения программы. Размерность для этих типов памяти может определяться соответственно переменными или выражениями. Значения этих переменных обязательно должны быть определены к моменту распределения памяти либо в операторе DECLARE, либо в операторе ALLOCATE.

Пример

```
XYZ: PROCEDURE;
     DECLARE A(100, 50), B(N) CONTROLLED;
       .
       .
       .
     GET LIST (I, J, N);
     ALLOCATE B(N);
     GET LIST (B);
```

```
      CALL ABC (A, B, I, J, N);
         .
         .
         .
      END XYZ;
ABC: PROCEDURE (C, D, L, M, N);
      DECLARE C(L, M), D(N);
         .
         .
         .
      END ABC;
```

В операторе DIMENSION вызванной процедуры звездочки показывают, что этот идентификатор имеет ту же размерность или длину, что и соответствующий идентификатор в вызывающем блоке. Звездочки неприменимы для переменных BASED. Оператор DIMENSION в XYZ (см. ранее приведенный пример) можно было бы записать так:

```
DECLARE A (100, 50), B(*) CONTROLLED;
```

Пример

```
DECLARE A (100, 50);
CALL CBA(A);
    .
    .
    .
CBA: PROCEDURE (X);
      DECLARE X(*, *);
            .
            .
            .
```

В этом примере граничные значения размерности для переменной X будут передаваться, когда будет вызвана процедура CBA.

Определить размерность массива можно с помощью встроенной функции DIM. Форма ее записи такова: DIM (арг1, арг2), где арг 1 представляет собой имя массива, размерность которого нужно определить , а арг2 — определяемое граничное значение размерности. Значение арг2—двоичное целое число, а если оно дано не в этой форме, то производится его преобразование. В ранее приведенном примере встроенная функция DIM (A, 2) даст значение 50. (В подмножестве ПЛ/1 употребление встроенной функции DIM запрещено). В операторе ALLOCATE также можно употребить звездочку для указания длины строки или размерности переменной.

Пример

```
DECLARE C CHARACTER (5) VARYING CONTROLLED;
            .
            .
            .
ALLOCATE C CHARACTER (5);
            .
            .
            .
ALLOCATE C CHARACTER(L);
            .
            .
            .
ALLOCATE C CHARACTER (*);
```

Первый оператор ALLOCATE установит максимальную длину строки символов, равную 5, второй — L, а третьей, где стоит звездочка, снова — L (обращаясь к самому последнему перед этим распределению).

6.7. РЕКУРСИВНЫЙ ВЫЗОВ ПРОЦЕДУР

(В подмножестве ПЛ/1 рекурсивный вызов процедур не разрешается.) До сих пор рассматривались способы вызова только неактивных процедур. Однако некоторые процедуры во время вызова могут быть и активными. Активная процедура может быть еще раз вызвана (рекурсивно) при условии, что она определена описателем RECURSIVE. Это делается необязательно путем включения в оператор PROCEDURE описателя RECURSIVE;

Например,

```
ABC: PROCEDURE RECURSIVE;
```

Область действия этого описателя включает также все вторичные точки входа процедуры ABC.

Всякий раз, когда вызывается рекурсивная процедура, значения переменных, определяемых всеми нестатическими идентификаторами, «опускаются» в магазин так же, как «опускаются» управляемые переменные. Когда выполнение процедуры при данном обращении заканчивается, эти значения «поднимаются» из магазина так же, как значения управляемых переменных. Разбирая следующий пример, уясните эти правила. Далее приводится процедура-функция для вычисления $n! = n\ (n-1)\ \ldots\ (1)$, где n — положительное целое число. Другой способ определения $n!$:

$$n! = \begin{cases} 1, & \text{если } n=1 \text{ или } n=0 \\ n\,(n-1)!, & \text{если } n>1 \end{cases}$$

```
1.  FACT: RROCEDURF(N) FIXED (10) RECURSIVE;
2.        DECLARE M FIXED (10), FACT RETURNS (FIXED(10));
3.        IF N=1 | N=0 THEN RETURN(1);
4         M=FACT(N-1)*N;
5.        RETURN(M);
6.        END FACT;
```

Процедура, которая первоначально вызывает FACT, должна иметь вид FACT RETURNS (FIXED (10)) в операторе DECLARE, поскольку формат вычисленного значения отличается от формата, определенного по умолчанию. Кроме того, сама функция FACT должна содержать такое описание, так как она сама может обратиться к себе.

В дальнейшем последовательные распределения памяти для определенной переменной будут обозначаться штрихами. Предположим, что эта процедура была вызвана при значении N = 3.

Первый вызов. При первом обращении из внешней процедуры N′ имеет значение 3 и место в памяти определено для M′. Поскольку N′ равно 3, то управление передается оператору (4), и оператор (4) вычисляет N′ — 1 = 2 и вызывает процедуру FACT второй раз.

Второй вызов. N″ получает значение 2, в то время как старое значение «опущено» в магазин. Место в памяти определено для M″, M′ «опускается». Поскольку N″ равно 2, управление передается оператору (4), и оператор (4) вычисляет N″ — 1 = 1 и вызывает процедуру FACT в третий раз.

Третий вызов. N‴ получает значение 1, а его старое значение (2) «опускается» в магазин. Место в памяти определено для M‴, а M″ «опускается». Поскольку N‴ равно 1, то выполняется конструкция THEN оператора (3), заканчивая, таким образом, третий вызов процедуры FACT, освобождая N‴ и M‴ и передавая значение 1 второму вызову процедуры FACT. Значения N″ и M″ «поднимаются». N″ имеет значение 2.

Выполняется оператор (4), вычисляя значение 1* 2 = 2, которое записывается в M″. Следующим выполняется (первый раз) оператор (5), заканчивая второй вызов процедуры FACT, освобождая N″ и M″ и передавая значение 2 в точку вызова, которым является оператор (4). Значения N′ и M′ «поднимаются» из магазина. N′ имеет значение 3. Оператор (4) выполняется еще раз, вычисляя значение 2* 3 = 6, которое записывается в M′. Следующим выполняется (второй раз) оператор (5), заканчивая первый вызов процедуры FACT, освобождая N′ и M′ и передавая требуемый результат 6 в первоначальную точку вызова, которая находилась в другом блоке.

В этом примере можно было бы опустить оператор (2), а значение M в операторах (4) и (5) заменить на N. Конечный результат был бы тем же самым и дополнительная память не потребовалась бы.

Полезно проследить за логикой следующей программы тем же способом, которым был разобран ранее приведенный пример:

```
                /* CHAPTER #6—EXAMPLE #1 */
                /* N FACTORIAL */

1  E601: PROCEDURE OPTIONS (MAIN);
2        DCL FACT RETURNS (FIXED(10));
3        DO N=0 TO 7;
4        PUT SKIP EDIT (N, FACT(N)) (F(5), F(20));
5        END; END; E601;

1  (CHECK (M, N));
   FACT: PROCEDURE (N, FIXED (10) RECURSIVE;

2       DCL FACT RETURNS FIXED(10));
3          DCL M FIXED (10);
4          IF N=1 | N=0 THEN RETURN(1);
6          M=FACT (N-1)*N;
7          RETURN(M);
8          END FACT;
```

```
       0                1
       1                1
       2
M=             2;
                  2
       3
M=             2;
M=             6;
                  6
       4
```

```
M=          2;
M=          6;
M=         24;
                    24
      5
M=          2;
M=          6;
M=         24;
M=        120;
                   120
      6
M=          2;
M=          6;
M=         24;
M=        120;
M=        720;
                   720
      7
M=          2;
M=          6;
M=         24;
M=        120;
M=        720;
M=       5040;
                  5040
```

Коммеитарий к программе

```
/* Глава 6, пример 1*/
/*N  факториал  */
```

Нужно заметить, что ценность приведенного примера ограничена, так как 141 дает в результате одиннадцать цифр. Более полезной была бы функция с плавающей точкой. Подобная логика возможна и в том случае, если последовательные вызовы относятся не только к одной и той же, но и к некоторой другой активной процедуре. Идентификаторы с описателем STATIC не «опускаются» в магазин рекурсивными вызовами, а остаются доступными в течение всего времени выполнения программы.

6.8. УПРАЖНЕНИЯ

Короткие упражиеиия

1. На карте отперфорироваиы значения 3b 1. b2. b3. (b озиачает пробел). Каждая из приведеииых далее программ предусматривает применение карт.

Если программа правильиа, то что будет иапечатано? Если оиа неправильиа, то почему?

а)
```
ABC: PROCEDURE OPTIONS (MAIN);
     GET LIST (N);
     DCL A (N);
     GET LIST (A);
     PUT LIST (N, A);
     END;
```

б)
```
ABC: PROCEDURE OPTIONS (MAIN);
     DCL A (N);
     GET LIST (N);
     GET LIST (A);
     PUT LIST (N, A);
     END;
```

в)
```
ABG: PROCEDURE OPTIONS (MAIN);
     DCL A (N) CONTROLLED;
     GET LIST (N);
     ALLOCATE A;
     GET LIST (A);
     PUT LIST (N, A);
    END;
```

г) то же, что в примере (в), только оператор DECLARE имеет форму DCL A (*) CONTROLLED;

```
д) ABC: PROCEDURE OPTIONS (MAIN);
        GET LIST (N);
        CALL XYZ;
        XYZ: PROCEDURE; ]
             DCL A (N);
             GET LIST (A);
             RETURN;
             END;
        PUT LIST (N, A);
        END;
е) ABC: PROCEDURE OPTIONS (MAIN);
        GET LIST (N);
        CALL XYZ;
        XYZ: PROCEDURE;
        DCL A (N);
        GET LIST (A);
        PUT LIST (N, A);
        RETURN;
        END;
   END;
ж) ABC: PROCEDURE OPTIONS (MAIN);
        GET LIST (N);
        CALL XYZ (N);
        PUT LIST (N);
        END;
   XYZ: PROCEDURE (N);
        DCL A (N);
        GET LIST (A);
        PUT LIST (A);
        RETURN;
        END;
```

2. Предложеиные задачи относятся к структуре программы, приведеиной на рис. 6.4.

а) составьте таблицу, расположив слева обозиачеиия каждого блока, а справа укажите, какие процедуры могут быть вызваиы этим блоком;

б) процедура C вызвала процедуру G, а в процедуре G имеется оператор GO TO LAB; (где LAB — константа типа метки). Объясиите, где еще мог бы быть оператор с меткой LAB;

в) если процедуру G нужио вызвать поцедурой F как процедуру-фуикцию (без фактическнх параметров), то какие добавочные операторы иужно добавить к программе?

г) какова область действия перемеиной Y?

д) какова область действия перемеиной X?

е) во время выполиеиия программы производились следующие последовательные обращеиня и выходы из процедур: CALL B, CALL C, RETURN, CALL E, CALL C. Программа достигла оператора CALL E, который входит в блок BEGIN D. Возможио ли это?

ж) то же, что в примере (е), за исключеиием того, что оператором, который иужио выполиить следующим, является CALL G;

з) выполиение программы достнгло иекоторого оператора в процедуре A или в одном из блоков, входящих в A. Необходимо вызвать процедуру H. Возможно ли это? Если иевозможио, объясиите, можно ли что-иибудь предприиять, чтобы это стало возможным?

н) пусть оператор процедуры для процедуры H записаи в виде: H : PROCEDURE (M, N); В процедуре имеется оператор RETURN (4* M—N). Можио ли вызвать процедуру H оператором процедуры G, имеющим форму: K = 2* H (I, J); (см. рис. 6.4)?

3. Что будет напечатано после обработки следующей программы?

```
XYZ: PROCEDURE OPTIONS (MAIN);
     DCL K STATIC EXTERNAL INITIAL (0);
     X=5;
     PUT LIST (K, FOF (X), K);
     END XYZ;
FOF: PROCEDURE (X) RECURSIVE;
     DCL K STATIC EXTERNAL INITIAL (0);
     K=K+1;
     IF X=0 THEN RETURN (0);
     RETURN (X+GOF (X));
     END FOF;
GOF: PROCEDURE (X) RECURSIVE;
     RETURN (4.-FOF (X-1.));
     END GOF;
```

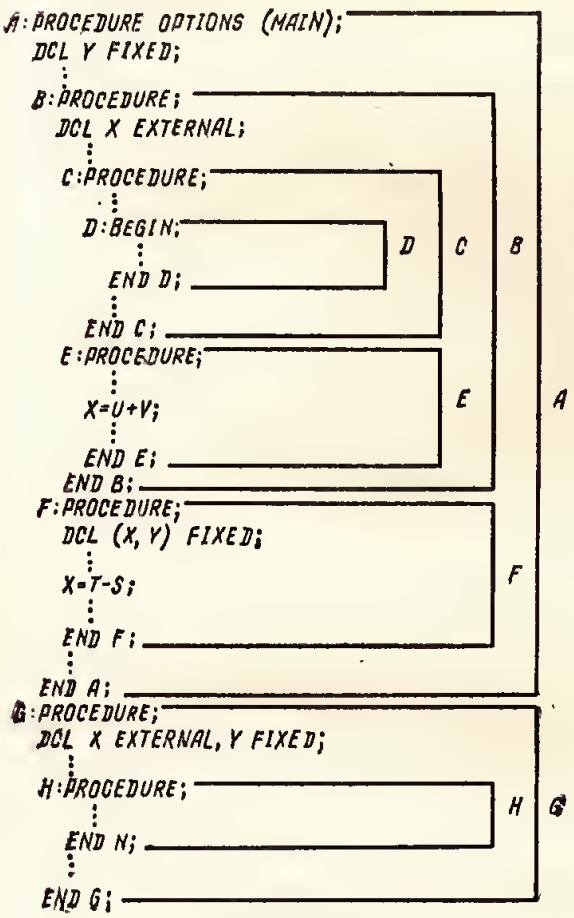
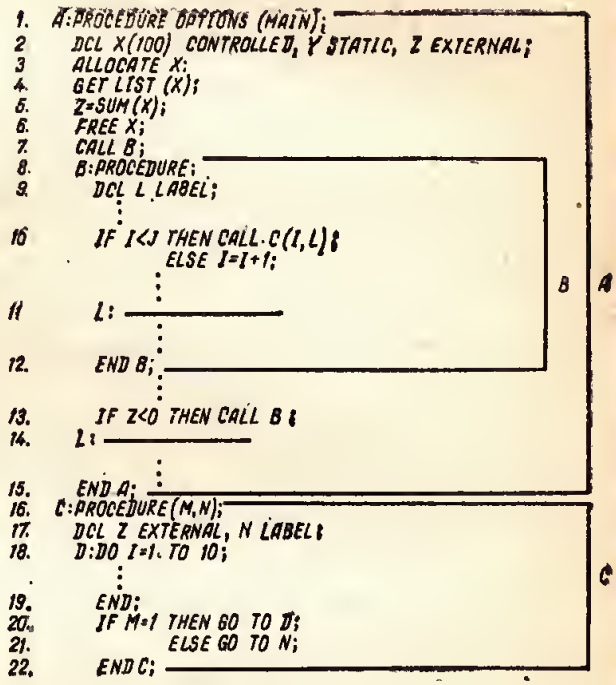


Рис. 6.4.



Рнс. 6.5.

4. Аиализируя структуру программы, приведениой на рнс. 6.5, объясните область действня имен, способы актнвнзации н окончання блоков, распределенне и освобожденне памятн н, в общнх чертах, как эта программа действует (чнсла слева даиы только для ссылки).

Задачи для программирования

1. а) В примере Б параграфа 2.8 приведены две процедуры для решения квадратного уравнения. Используйте одну из них как внешнюю процедуру без употребления операторов GET илн PUT. Вы могли бы проверить правиль-

ность вашей программы при помощи записи простой главной процедуры для вызова этой внешней процедуры, а затем попытаться просчитать несколько контрольных задач;

б) напишите внешнюю процедуру для нахождения действительного корня кубического уравнения $ax^3 + bx^2 + cx + d = 0$ с помощью метода деления пополам (он описан далее). Как только действительный корень будет найден, разложите данное кубическое уравнение на линейные и квадратные множители и напишите обращения к внешней процедуре из пункта (а) данного упражнения для нахождения двух остальных корней.

Метод деления пополам предполагает, что известны два приближенных значения X : XL и XR, находящихся по обе стороны искомого корня R. Пусть $f(x) = ax^3 + bx^2 + cx + d$ и, скажем, XL < R < XR. f(XL) и f(XR) должны иметь противоположные знаки. Теперь можно с помощью следующего алгоритма найти значение R:

1) вычислите $f\left(\frac{XL + XR}{2}\right)$;

2) если f(XL) и значение, полученное в (1), имеют противоположные знаки, то $XR = \frac{XL + XR}{2}$; в противном случае $XL = \frac{XL + XR}{2}$;

3) если XL — XR < TOL, где TOL — заданная точность вычисления, то прекратите вычисления и воспользуйтесь либо XL, либо XR как значением R с толерантностью ± TOL. Если XL — XR > TOL, то перейдите к шагу (1) и повторите процесс;

в) для проверки правильности программ, составленных в соответствии с заданиями (а) и (б), напишите простую главную процедуру.

2. В некотором географическом районе наблюдалось, что вероятность изменения погоды на следующий день по сравнению с сегодняшним подчиняется таким законам: вероятность того, что погода изменится, равна 2/3, вероятность того, что погода остается неизменной, равна 1/3. Но если в течение трех или более дней удерживаются одинаковые погодные условия, то эта вероятность изменяется таким образом: вероятность того, что погода изменится, равна 5/6, вероятность того, что погода останется неизменной, равна 1/6.

Если изменение погоды на следующий день ожидается, то вероятность того, что на следующий день будет дождь, снег, ясная или облачная погода, можно выразить таблицей:

ПОГОДА НА СЛЕДУЮЩИЙ ДЕНЬ

Погода сегодня	Дождь	Снег	Ясно	Облачно
дождь		1/6	2/6	3/6
снег	2/6		1/6	3/6
ясно	2/6	1/6		3/6
облачно	2/6	2/6	2/6	

Напишите программу для считывания карты данных, где отперфорированы ДОЖДЬ, СНЕГ, ЯСНО или ОБЛАЧНО (что показывает погоду сегодня). После этих слов на карте отперфорированы два целых числа. Первое число показывает количество дней, в которые подряд удерживалась одинаковая погода. Второе число показывает количество дней (включая сегодняшний день), по которым нужно получить прогноз погоды. Программа должна смоделировать погоду

на это количество дней в соответствии с указанными законами и выдать на печать прогноз погоды на каждый из дней.

Эту систему можно смоделировать, бросая игральную кость всякий раз, когда нужно сделать выбор. Например, если сегодня ДОЖДЬ, а вчера его не было, то сначала надо бросить кость, чтобы выяснить, изменится ли завтра погода. Если выпадут числа 1, 2, 3 или 4, то это изменение произойдет, если 5 или 6 — изменений не ожидается. Если изменение ожидается, кость надо кинуть еще раз. Если выпадет число 1, то завтра будет СНЕГ; если 2 или 3 — завтра может быть ЯСНО, если выпадет 4, 5 или 6 — завтра может быть ОБЛАЧНО.

При написании программы ориентируйтесь на столько внутренних и внешних процедур, сколько их требуется. Одной из них будет, вероятно, процедура-функция, которая будет вычислять случайные числа 1, 2, 3, 4, 5 или 6, имитируя бросание игральной кости.

3. Напомним, что $n! = n\,(n-1)\,(n-2)\ldots 1$ для целого числа n. $0!$ считается равным 1.

Факториальные многочлены записываются в виде $n^{(r)} = n\,(n-1)\,(n-2)\ldots(n-r+1)$ и $n^{(0)} = 1$, где n и r — целые числа

Заметьте, что $n^{(1)} = n$ и $n^{(n)} = n!$ Из этого определения сразу же следует, что

$$n^{(r+1)} = (n-r)\,n^{(r)}. \qquad (1)$$

Число различных способов (независимо от порядка), когда r объектов можно выбрать из n объектов, определяется формулой:

$$C(n, r) = \frac{n^{(r)}}{r!}, \quad n \geqslant r \geqslant 0, \qquad (2)$$

например,

$$C(4, 2) = \frac{4^{(2)}}{2!} = \frac{4 \cdot 3}{2 \cdot 1} = 6.$$

Напишите внешнюю рекурсивную процедуру, которая вычислит $C(n, r)$ для данных n и r по уравнениям (1) и (2). Для того чтобы проверить эту процедуру, напишите простую главную процедуру для вычисления $C(n, r)$ для $n = 8$ и $r = 0, 1, 2, 3, 4, 5, 6, 7, 8$.

4. В упражнении 9 из главы 5 рассмотрена печать графика функции. Цель настоящего упражнения — написание более общей процедуры для выполнения этой задачи.

Напишите внешнюю процедуру, которая будет вычерчивать на одном листе графики до 9 функций ($y = f(x)$) для заданного интервала значений X. Дополнительно предусмотрите возможность печати до трех базисных линий (y = константа) и списка значений X на левой стороне графика. Далее приведены некоторые условия для программирования этой задачи:

а) для всех значений функций и базисных линий масштабы должны быть выбраны так, чтобы графики заняли максимум места на странице. Для выполнения этой задачи нужно написать отдельную процедуру;

б) при определении координат точек графика необходимо производить округление значений так, чтобы кривая была по возможности гладкой;

в) значения масштабных множителей должны печататься;

г) символы, обозначающие точки графиков различных функций и базисных линий, должны быть выбраны так, чтобы различные графики и базисные линии легко различались;

д) нужно определить требования к памяти (если все координаты вычисляются и хранятся в памяти, то может потребоваться избыточное количество памяти);

е) система по умолчанию может быть дополнена явными описаниями с тем, чтобы использовать целую страницу и обеспечить непрерывную печать графика при переходе с одной страницы на следующую;

ж) может оказаться более целесообразным формяровать и печатать целые строки, а ие применять SKIP (0) для последовательных контуров прн данном значеини X;

з) еслн два графика пересекутся, то в одной точке необходнмо определнть, какой символ нлн символы проставлять в точке пересечення.

5. а) напншнте внешнюю процедуру, которая передавала бы вызвавшей процедуре массив данных, представляющий перетасованную колоду нз 52 обычных игральиых карт. «Колода», передаваемая по разлнчным вызовам, должна быть сформирована случайным образом н не должна составляться по какому-лнбо зараиее обусловленному образцу по последующнм вызовам нлн по последующнм проходам программы. Например, «карты» могли бы хранпться в массиве нз 52 снмволов; затем элемеиты этого масснва переставлялнсь бы случайиым образом.

Порядок и число перестаиовок можно было бы определять с помощью геиератора случайных чнсел, который выбнрал бы начальные значення с помощью встроенной функцин TIME;

б) напишите программу для моделнрования раскладывання пасьянса N раз, как это опнсано далее. Значеиие N читается в исходных данных. В выходных данных для каждого пасьянса должны быть показаны карты и порядок, в котором они выпадают во время всей нгры, и должно быть указаио, сошелся ли пасьянс.

Правила игры. Сверху перетасованной колоды снимаются н откладываются две карты. Еслн этн две карты одниаковы по зиачеиню (напрнмер, два валета илн две семерки), то сиимаются н открываются следующие две карты, которые кладутся на две первые. Это продолжается до тех пор, пока две открытые карты будут разиымн по значеняю. Тогда снимается н открывается третья карта. Если из трех открытых теперь карт любые две одиниаковы, то на ннх кладут две карты, снятые с колоды. Это продолжается до тех пор, пока трн открытые карты ие выпадут разными. Тогда сиимается н открывается четвертая карта Описаиный процесс протекает прн одном огранпченнн — нельзя раскладывать больше, чем восемь кучек карг.

Глава 7

ДОПОЛНИТЕЛЬНЫЕ СВЕДЕНИЯ О ЯЗЫКЕ ПЛ/1

В следующих разделах рассматрнваются дополнительные сведения о языке ПЛ/1. Эти разделы тематическн между собой не связаны н поэтому читать их можно в любом порядке.

7.1. ОПРЕДЕЛЕНИЕ ПО СООТВЕТСТВИЮ И ПО СОВМЕЩЕНИЮ

В Фортране оператор EQUIVALENCE отводит двум переменным, которые находятся в одной и той же программе нлн подпрограмме, одну и ту же область памяти. В ПЛ/1 той же цели служнт оператор DEFINED. Напрнмер, оператор DECLARE X DEFINED Y; определнт одно и то же место в памятн для переменных X н Y. В этом операторе Y должно быть определено ранее лнбо явно, либо неявно.

При определенин по соответствню ранее определенному элементу данных прнсванвается другое имя. Соответствне может быть между масснвамн однпаковой размерности нли между одной частью некоторого масснва н массивом другой размерностн. Оператор

```
DCL X (5, 4) FIXED (5,1) INIT ((20)0),
    Y 5, 4) DEFINED X;
```

определнт Y и X как разлнчные имена переменных одного и того же массива. Опнсатель INITIAL можно нспользовать только при первоначальном определенни ячейкн памятн. (В подмножестве ПЛ/1 масснвы должны быть однпаковой размерности.)

Спецнальные индексы тнпа i SUB (где i — целое число) можно употреблять для переопределення только частн масснва. Индекс i SUB определяет i-й нндекс переменной, который должен быть переопределен. (В подмножестве ПЛ/1 индекс i SUB запрещен.) Например, оператор DCL X (30) INIT((30)0), Y (5) DEFINED X (1 SUB); переопределит ячейки X (1), X (2), X (3), X (4) и X (5) идентнфнкаторами

Y (1), Y (2), Y (3), Y (4), Y (5). Индекс 1 SUB определяет первый индекс переменной, которую нужно переопределить, а именно переменной Y. Значения 1 SUB от 1 до 5 определяются размерностью данной переменной.

Оператор DCL X (30), Y (5) DEFINED X (25 + 1SUB); переопределит ячейки X (26), ..., X (30) как Y (1), ..., Y (5).

Оператор DCL A (50, 30) B (50) DEFINED A (1SUB, 15); переопределит пятнадцатый столбец массива A как одномерный массив B, состоящий из 50 элементов.

Оператор DCL U (5, 10), V (10, 5) DEFINED U (2SUB, 1 SUB); обеспечит легкий способ «взгляда на массив» как на массив 5 × 10 или как на массив 10 × 5.

Оператор DCL W (15, 15), X (15) DEFINED W (1 SUB, 1 SUB); даст главным диагональным элементам массива W имя одномерного массива X. При необходимости размерность нового массива в приведенных примерах может быть уменьшена. Например, оператор DCL W (15, 15), X (13) DEFINED W (1 + 1SUB, 1 + 1SUB); присвоит главным диагональным элементам массива (исключая два угловых элемента) имя одномерного массива X. Для установления требуемого соответствия переменные i SUB могут быть проставлены в выражениях, как это было сделано в некоторых из ранее приведенных примеров.

Оператор DCL X(5, —2: 2), Y DEFINED X (3, 0); определяет, что идентификатор X (3,0) и Y определяют одну и ту же переменную.

При употреблении описателя DEFINED описателем INITIAL можно пользоваться только при первоначальном определении области памяти, а описатель VARYING вообще неприменим. Элемент, определенный DEFINED, всегда считается INTERNAL, хотя первоначально идентификатор мог иметь описатель EXTERNAL. Кроме того, нельзя переопределять переменную в терминах другой переопределяемой переменной.

В ситуациях, которые приведены далее, нужно быть особенно осторожным:

```
A: PROCEDURE;
   DCL X (10) EXTERNAL, Y (10) DEFINED X (1SUB);
        .
        .
   END A;
B: PROCEDURE;
   DCL X (10) EXTERNAL;
        .
        .
   END B;
```

В этом примере массив X определен как в процедуре A, так и в процедуре B. С другой стороны, переменная Y определена только в процедуре A. Если процедура A вызывает процедуру B и процедура B изменяет значения элементов массива X, то последующие обращения к Y после передачи значений из B в A приведут также к изменению значений элементов массива Y.

Определение по совмещению позволяет совмещать память при переопределении. Например, оператор DCL A CHAR (10), B CHAR (3) DEFINED A; определяет, что строка символов B совмещается первыми тремя элементами строки символов A, длина которой равна 10 .

Если нужно, чтобы совмещение начиналось не с первого элемента, а с некоторого другого, то можно употребить специальный описатель POSITION (d), где d — целое число. Этот описатель применяется только с переменными BIT или CHARACTER.

Оператор DCL A CHAR (10), B CHAR (3) DEFINED A POSITION (6); определяет, что строка символов B совмещается с шестым, седьмым и восьмым элементами строки A.

Нужно очень тщательно следить за тем, чтобы определяемая переменная не вышла за пределы поля, которое переопределяется.

DEF — допустимое сокращение от DEFINED. (В подмножестве ПЛ/1 такое сокращение запрещено, а употребление описателя POSITION не допускается.)

7.2. ПРЕРЫВАНИЯ ПРОГРАММЫ

Ошибки в синтаксисе, которые обнаруживаются во время трансляции большинством трансляторов, понятны и их не слишком трудно исправить. Но ошибки, которые обнаруживаются уже во время выполнения программы, принадлежат к наиболее трудно исправляемым; чаще всего в этом случае выполнение программы просто прекращается, и программист получает сообщение об ошибке, но о причине ошибки ему почти ничего не сообщается.

Транслятор ПЛ/1 располагает рядом возможностей проверки программы на наличие ошибок во время ее выполнения.

Некоторые из них находятся под контролем программиста, в том смысле, что если программист хочет, он может сам определить, можно ли игнорировать эти ошибки, а если нельзя, то выработать действия, которые необходимо предпринять при возникновении ошибки. Транслятор ПЛ/1 может во время трансляции вставить в транслируемую программу специальные подпрограммы, обеспечивающие управление в определенных ситуациях либо по умолчанию, либо по указанию программиста. Диапазон этих ситуаций (назовем их *разрешающими*) очень широк, начиная от проверки переполнения порядка и кончая контролем последней строки на текущей странице.

При такой разрешающей ситуации возникает либо стандартная реакция системы, определяемая по умолчанию, либо создается реакция, специально определенная программистом.

Мы уже встречались с такой контролируемой программистом ситуацией в операторе

ON ENDFILE (имя файла) оператор;

Выполнение этого оператора приводит к возникновению разрешающей ситуации ENDFILE. При попытке считать что-либо после метки конца указанного файла возникает разрешающая ситуация

ENDFILE и выполняется написанный программистом оператор, стоящий после скобок с именем файла. При стандартной реакции системы печатается сообщение об ошибке, и выполнение программы прекращается.

Таблица ситуаций с применением оператора ON дана в приложении Г. В таблице показано также, для каких операторов ON программист сам может создать реакцию системы или маскировать выполнение разрешающей ситуации, а в каких случаях возникает стандартная реакция системы, определяемая по умолчанию.

Имена ситуаций. Имя ситуации может быть указано перед оператором ПЛ/1 и его метками. Общий вид имени ситуации:

```
(имя ситуации [, имя ситуации] ...):
```

Например, запись

```
(SUBSCRIPTRANGE) : ABC : A (I) = B (I+1);
```

вызовет сравнение значения индекса I с граничными значениями. В случае выхода за пределы допустимых границ возникает разрешающая ситуация.

Реакция на ситуацию может быть замаскирована путем добавления слова NO к имени ситуации в качестве приставки. Тогда запись может, например, выглядеть так:

```
(NOOVERFLOW) : X = Y + Z;
```

Имя ситуации может стоять перед любым оператором ПЛ/1, за исключением операторов DECLARE или ENTRY.

Область действия имени ситуации определяется по следующим правилам:

Оператор с записанным перед ним именем ситуации	Область действия распространяется
Оператор присваивания	Только на данный оператор, но ни на одну из процедур, которую этот оператор вызывает
Оператор IF	На выражение, следующее за IF. Перед конструкциями THEN и ELSE могут быть свои собственные имена ситуаций
ON	Не применяется к ON-unit (будет рассмотрен в этом параграфе)
DO	На выражения в самом операторе DO, но не на операторы группы DO
PROCEDURE	На все операторы вплоть до оператора END, включая все вложенные блоки. На вызываемые процедуры, лежащие вне блока, действия не оказывает
BEGIN	Аналогично оператору PROCEDURE

Пример

```
(NOFIXEDOVERFLOW, SUBSCRIPTRANGE):ABC:PROCEDURE OPTIONS (MAIN);
                                    .
                                    .
                                    .
                (FIXEDOVERFLOW):M=1*J;
                                    .
                                    .
                                    .
```

```
(N OSUBSCRIPTRANGE): XYZ:PROCE DURE;
                    :
                    :
                END XYZ;
                END ABC;
```

В данном примере ситуация FIXEDOVERFLOW (переполнение с фиксированной точкой) будет маскирована для всех операторов процедур ABC и XYZ, за исключением оператора M = I* J; ситуация SUBSCRIPTRANGE (выход значения индекса за допустимые границы) будет проверяться для всех операторов ABC (с организацией соответствующей стандартной реакции), но будет маскирована при любых обращениях к процедуре XYZ.

Оператор ON CONDITION. Ранее было показано, какими способами программист может создавать разрешающие ситуации и маскировать их. Всякий раз, когда возникает разрешающая ситуация, действует стандартная реакция системы, если только программист не хочет организовать свою собственную реакцию с помощью оператора ON.

Оператор ON обычно имеет вид:

```
ON имя ситуации [SNAP] {ON-unit }
                       {SYSTEM;}
```

(В подмножестве ПЛ/1 в качестве ON-unit может служить только оператор GO TO или пустой оператор.)

Если записано слово SYSTEM (за ним следует точка с запятой), то возникает стандартная реакция системы. Слово SYSTEM употребляется для того, чтобы замаскировать действие предшествующего оператора ON и по умолчанию передать управление системе. ON-unit — это подпрограмма, написанная программистом, определяющая реакцию на ошибку. Правила написания ON-unit будут рассмотрены далее.

Запись необязательного слова SNAP позволяет печатать информацию, определяемую системой, если разрешающая ситуация возникает до выполнения ON-unit или до реакции системы, определяемой по умолчанию. (В подмножестве ПЛ/1 употребление слова SNAP запрещено.)

ON-unit может состоять из одиночного оператора ПЛ/1 без метки или блока BEGIN без метки. Операторы IF, ON, RETURN, FORMAT, DECLARE, DO, PROCEDURE и END не могут быть в ON-unit. Блок BEGIN в ON-unit может включать любой оператор ПЛ/1, кроме RETURN. Но в блок BEGIN могут быть вложены другие блоки, которые включают операторы RETURN.

Когда возникает ситуация, определенная в операторе ON, выполняется ON-unit и управление передается на оператор, который вызвал появление ошибки. ON-unit обрабатывается как вызванная процедура, даже если он состоит из одного оператора. Как и при обращении к процедуре, может быть выполнен оператор GO TO, независимо от того, является ли он одиночным оператором или входит в блок BEGIN. Он передаст управление некоторому оператору, но только не тому, который вызвал появление ошибки.

Если необходимо, можно прибегнуть к пустому ON-unit. Тогда за именем ситуации или словом SNAP просто ставится точка с запятой.

В этом случае при возникновении разрешающей ситуации никакой реакции системы не происходит. Запись пустого ON-unit со словом SNAP — один из эффективных способов маскирования ситуации, которую каким-либо другим способом маскировать нельзя. Это дает дополнительную возможность отмечать прерывания. Результат выполнения оператора ON определен в блоке, в котором он находится, и во всех блоках, активизированных этим блоком. Его действие прекращается выполнением оператора с тем же именем ситуации, что и в ранее указанном операторе ON или в операторе REVERT (этот оператор будет рассмотрен далее).

Если результат действия оператора ON перекрывается действием другого оператора ON в вызванной процедуре, результат действия первоначального оператора ON восстанавливается с окончанием вызванной процедуры. Например,

```
ABC: PROCEDURE OPTIONS (MAIN);
          .
          .
     ON OVERFLOW GO TO L;
          .
          .
     CALL XYZ;
XYZ: PROCEDURE;
          .
          .
     ON OVERFLOW GO TO LL;
          .
          .
     END XYZ;
          .
          .
     END ABC;
```

При выполнении программы этого примера все переполнения, происходящие в процедуре ABC до появления оператора ON, вызывают стандартную реакцию системы.

Переполнение, происшедшее после выполнения оператора ON, вызывает передачу управления оператору процедуры ABC с меткой L. Это положение сохраняется до выполнения оператора ON вызванной процедуры XYZ. С этого момента и до тех пор, пока не закончится выполнение процедуры XYZ, переполнение будет приводить к передаче управления оператору LL, который может находиться либо в процедуре ABC, либо в процедуре XYZ. После окончания процедуры XYZ действие оператора ON OVERFLOW GO TO L; восстанавливается до окончания выполнения программы.

Оператор REVERT (Отменить). С помощью оператора REVERT можно отменить действие последнего выполняемого оператора ON. Оператор REVERT имеет вид:

```
REVERT имя ситуации;
```

Этот оператор действует только на тот блок, в котором он записан.

Если в блоке выполняется оператор REVERT, то оператор ON, имеющий то же имя ситуации, что и REVERT, и выполняющийся после оператора REVERT, рассматривается как пустой оператор.

Оператор SIGNAL (Сигнал). Оператор SIGNAL служит для моделирования указанной в нем ситуации. Он часто используется при проверке реакции программы на ошибки. Его общий вид:

```
SIGNAL имя ситуации;
```

Если во время выполнения оператора SIGNAL имя ситуации маскируется, то он рассматривается как пустой оператор.

Ситуация CONDITION (Условие). (В подмножестве ПЛ/1 употребление ситуации CONDITION запрещено.) С помощью оператора CONDITION программист может ввести свои собственные имена ситуаций. Общая форма такой записи:

```
CONDITION (идентификатор)
```

Например, ON-unit можно записать следующим образом:

```
ON CONDITION (WRONG) BEGIN; ... END;
```

Последующее выполнение оператора SIGNAL CONDITION (WRONG); приведет к вызову ситуации CONDITION и будет выполняться блок BEGIN.

Ситуация CONDITION может быть вызвана только оператором SIGNAL.

Ситуация CHECK (Проверка). (В подмножестве ПЛ/1 употребление ситуации CHECK запрещено.) Этот оператор уже рассматривался в главе 2. В качестве префикса он может появиться только либо в операторе PROCEDURE, либо в операторе BEGIN. Из следующего примера видно, как CHECK может использоваться в операторе ON:

```
ON CHECK(Z) PUT DATA (A  B, C, X, Y, Z);
```

В таких случаях текущие значения A, B, C, X, Y и Z могут быть напечатаны только тогда, когда будет вычислено новое значение Z. Для того чтобы обеспечить вывод данных с метками, необходим вывод, управляемый данными. Такое действие отличается от действия, вызванного употреблением CHECK в качестве префикса

```
(CHECK (A, B, C, X, Y, Z));
```

в операторе PROCEDURE. В этом последнем случае он вызывает прерывание в выполнении программы всякий раз, когда изменяется один из идентификаторов.

Употребление ON-unit может привести совсем к другому результату. Например, следующая запись могла бы быть очень полезной при отладке программы, так как она выдала бы на печать значения переменных только в том случае, когда $Z = 0$;

```
ON CHECK (Z) BEGIN; IF Z=0 THEN PUT DATA (A, B, C, X, Y, Z); END;
```

Ситуации-функции. Некоторые встроенные функции могут помочь программисту обнаружить причину возникновения разрешающей ситуации. Иногда такие функции помогают исправить ошибки и продолжать выполнение программы. Эти функции не содержат фактических параметров, их можно применять только в ON-unit или процедуре, которую этот ON-unit вызывает.

Все эти функции, вместе с другими встроенными функциями ПЛ/1, даны в приложении Д.

7.3. ШАБЛОНЫ

Шаблоны предназначены в первую очередь для редактирования данных ввода-вывода. Спецификацнн шаблонов задаются описателем PICTURE оператора DECLARE или форматом P. В Фортране такнх средств нет. Но программисту, знакомому с Коболом, шаблоны ПЛ/1 покажутся знакомыми, так как в Коболе они используются интенсивно. Шаблоны делятся на два основных класса, описывающие символы и опнсывающне чнсла. Все шаблоны как в описателе PICTURE, так н в формате P заключаются в одинарные кавычкн.

Шаблоны для описания символов. Шаблоны для опнсання снмволов очень удобны для опнсания символьных форматов. Единственными разрешенными снмволами для этого типа спецификации являются снмволы X, A и 9. В шаблоне должен быть по крайней мере однн снмвол X илн A.

Символы X, A н 9 имеют следующне значения:

X означает, что соответствующее поле может содержать любой символ;

A означает, что соответствующее поле может содержать любую букву алфавита нли пробел;

9 означает, что соответствующее поле может содержать любую десятичную цифру или пробел. (В подмножестве ПЛ/1 разрешается только спецификация X.)

Пример

Строка снмволов 'Xb = bY/14.6' (b означает пробел) состоит из 10 снмволов. Шаблон для описанння этой строки будет состоять из 10 символов X, A или 9, заключенных в кавычки. Существует несколько возможных вариантов. Следующая таблица дает представление о таких вариантах:

Символ в данной строке	Шаблон для опнсания символов
X	X или A
b	X, A или 9
=	X
b	X, A или 9
Y	X или A
/	X
1	X или 9
4	X или 9
.	X
6	X или 9

Следовательно, существует 2 · 3 · 1 · 3 · 2 · 1 · 2 · 2 · 1 · 2, или 288 вариантов опнсання шаблонами данной строки снмволов, напрнмер 'XXXXXXXXXX', 'AAX9AX99X9' н 'X9X9XX99X9'. (Однако нз-за не точно определенной строки символов с различнымн специфнкациями могут возникнуть ошибки.) Строки символов левоустановленные, т. е. их запись всегда производнтся слева направо. Если необходимая

ширина поля не соответствует заданному размеру, то запись пробелов или отбрасывание символов производится справа.

Шаблоны для описания чисел. Эти шаблоны, как показывает само название, служат только для описания числовых форматов. В шаблонах для описания чисел разрешается гораздо большее разнообразие символов, чем в шаблонах для описания строк символов. Но в них нельзя употреблять символы X и A. Значения, присваиваемые числовым полям, всегда связаны с десятичной точкой, и запись пробелов или отбрасывание символов производится либо справа, либо слева, если необходимая ширина поля не соответствует заданному размеру. Далее будут рассмотрены символы для спецификаций числовых шаблонов и приведены примеры. Данные для каждого примера расположены в следующем порядке: исходные данные, спецификация шаблона, значение строки символов.

Символ 9 означает, что соответствующее поле может содержать любую десятичную цифру. (Заметьте, что символ 9 в шаблоне для описания символов показывает, что поле может содержать пробел. В шаблоне для описания чисел возможность употребления пробела в этом случае исключается.)

Примеры

```
1234 — '9999' — '1234'
1234 — '99' — '34'
```

Символ определяет место десятичной или двоичной точки. Этот символ в шаблоне числа может быть записан только один раз, и если он отсутствует, то считается, что подразумеваемая точка всегда находится справа. Символ V делит поле на два подполя — справа и слева от подразумеваемой точки. Значение строки символов точку не содержит.

Примеры

```
1234 — '9999V' — '1234'
1234 — 'V9999' — '0000'
12.34 — '99V99' — '1234'
12.34 — '9V9' — '23'
```

Символ Z означает, что соответствующее поле данных содержит условную цифру. Если эта цифра равна нулю, то она заменяется пробелом; если она не равна нулю, то она остается неизменной. Символ Z может находиться справа от символов 9, T, I, R или от плавающих символов (см. далее). Символом Z нельзя пользоваться в одном и том же подполе с символом*. С помощью символа Z в некоторых случаях устраняют нули, стоящие перед числом.

Примеры

```
1234 — 'ZZZ9' — '1234'
0123 — 'ZZZ9' — 'b 123'
0010 — 'ZZZZ' — 'bb10'
01.23 — 'Z9V99 — ' b 123'
```

Символ* аналогичен символу Z, но им заменяют нули, стоящие перед числом. Символ* нельзя записывать в одном и том же подполе с символом Z. (Этот символ применяется для замены нулей, стоящих перед числом при печатании чеков и других финансовых документов.)

Примеры

```
1234 — '****' — '1234'
0123 — '***9' — '*123'
0010 — '****' — '**10'
01.23 — '*9V99' — '*123'
```

Если в спецификации после символа V употребляется символ Z или символ*, то в обоих подполях должен стоять один и тот же символ: либо Z, либо*. Нули в начале дробной части числа не устраняются, если после десятичной точки есть какие-нибудь значащие цифры.

Примеры

```
12.03—'** V **'—'1203'
00.01—'** V **'—'** 01'
```

Символ Y означает, что соответствующее поле данных содержит условную цифру. Если эта цифра равна 0, то она заменяется пробелом. Если цифра ненулевая, она не меняется. Символ Y заменяет все нули, где бы они ни стояли.

Примеры

```
0100—'YYYY'—' b1bb'
1020—'9YY9'—'1b20'
```

Символы (,), (.), (/) и b (пропуск) вносятся в поле данных в качестве редактирующих символов. Запятая, точка и косая черта — условные символы в том смысле, что они могут представлять соответствующий символ в строке символов, полученной в результате, а могут и не представлять его. Символ b всегда означает введение пробела. Если подполе, содержащее запятую, косую черту или точку, содержит, кроме того, символы подавления нулей (Z, Y или *), то эти символы будут вставляться в соответствующую позицию только в том случае, если слева от них имеется некоторая значащая цифра. Если значащих цифр слева нет, то Z и Y заменяются пробелом, а * остается. Если запятая или точка находится внутри строки плавающих символов (см. далее), то считается, что строка, содержащая плавающий символ, включает запятую, косую черту или точку, которые присваиваются плавающему символу. Точка, стоящая в шаблоне, не определяет десятичную точку в строке результата. Это делает только символ V. (В подмножестве ПЛ/1 употребление косой черты запрещено.)

Примеры

```
1234—'99,99'—'12,34'
1234.56—'9,999V.99'—'1,234.56'
00.01—'ZZV.ZZ'—'bb.01'
00123.45—'**,999V.99'—'***123.45'
9845—'9/9/99'—'9/8/45'
1234—'99BB9B9'—'12bb3b4'
```

Символ S означает, что соответствующее поле должно содержать знак плюс, если число в поле положительное или нуль, и знак минус, если число в поле отрицательное.

Символ + означает, что соответствующее поле должно содержать знак плюс, если число в поле положительное или нуль, и пробел, если число в поле отрицательное.

Символ — означает, что соответствующее поле должно содержать знак минус, если число в поле отрицательное, и пробел, если число в поле положительное или нуль.

Символ $ означает, что соответствующее поле содержит знак доллара. Он может размещаться либо в крайней левой, либо в крайней правой части подполя.

Плавающие символы. Если символы S, +, — или $ записаны в шаблоне только один раз, то выполняются описанные выше првила. Если некоторые из этих символов проставлены подряд, то это означает, что символ «сдвигается» и проставляется перед первой значащей цифрой строки результата.

Примеры

```
12.34—' $ 99V.99'—' $ 12.34'
01 23—' $ Z9V.99'—' $ b1.23'
01.23—' $$9V.99'—'b $ 1.23'
123—' $$$. $ 99'—' bbb $ 123'
—123—' S 999'—'—123'
—123.45—'+99 V.99'—' b 123.45'
```

Символы CR и DB, означающие кредит и дебет, определяют, что если значение данных отрицательное, то соответствующее поле содержит буквы CR или DB; в обратном случае в поле вставляются пробелы. Эти символы применимы только с данными REAL и печатаются в крайней правой позиции.

Примеры

```
—12.34—'$ ZZV. 99 CR'—' $ 12.34 CR'
—12.34—' $ ZZV.99DB'—' $ 12.34 DB'
 12.34—' $ ZZV.99DB'—' $ 12.34bb'
```

Символы T, I и R означают, что соответствующее поле исходных данных содержит цифру со знаком, отперфорированным в поле этой цифры. Они не могут быть в подполе, которое содержит символы шаблонов S, + или —. Если взят символ T, то цифра и знак перфорируются в одном поле. При символе I цифра и знак перфорируются в одном поле, если отперфорированная величина положительная или равна нулю. Если берется символ R, то цифра и знак перфорируются в одном поле только для отрицательных величин. Пробивка 11 означает знак — , а пробивка 12 означает знак +.

Примеры

```
123.—'99T'—'12C'
—123.—'99T'—'12L'
```

Символ K означает, что в поле справа от этого символа записан показатель степени числа. Этот символ не соответствует никакому символу поля.

Символ E означает место буквы E в числовых данных и указывает в поле начало показателя степени.

Примеры

```
.1234E05—'V.9999E99'—'.1234E05'
—12.34E—05—' S99V.99KS99'—'—12.34—05'
```

Символ F указывает на наличие в шаблоне масштабного множителя для данных с фиксированной точкой. Символ F записывается в крайней правой позиции подполя; за ним может следовать десятичная целая константа n со знаком или без него, заключенная в круглые скобки.

Если масштабный множитель положительный, то константа означает, что десятичная точка в исходных данных сдвинута вправо на n

позиций. Если число n отрицательное, то десятичная точка в исходных данных сдвинута на n позиций влево.

Примеры

```
1230—'999F(1)'—'123'
.12—'99 F (—2)'—'12'
```

Коэффициенты повторения. Для повторения символа в шаблоне можно пользоваться коэффициентом повторения, который записывается как десятичное число в круглых скобках. Например, запись '(5) 9 V. 99' эквивалентна записи '99999V. 99'.

P-формат. P-формат может быть полезен при редактировании данных во время ввода-вывода. Форма записи имеет вид:

```
P'спецификация шаблона'
```

где спецификацией будет один из описанных символов. При выводе данных эта спецификация определяет, каким образом выполняется редактирование элемента данных до вывода. При вводе данных эта спецификация определяет способ преобразования данных во внутреннее представление. (В подмножестве ПЛ/1 P-формат запрещен.)

Описатель PICTURE. Описатель PICTURE служит для описания внутренней и внешней форм представления полей чисел или строк символов и для редактирования данных. Его общая форма имеет вид:

```
PICTURE ('спецификация'),
```

где в скобки заключена спецификация шаблона для описания символьных или числовых данных. Слово PICTURE может быть сокращено до PIC. (В подмножестве ПЛ/1 такое сокращение не допускается.)

Спецификация может определять описатели BASE, SCALE и точность. Например, DCL A PICTURE 'XAA999'; определяет, что A есть строка символов из 6 элементов, а DCL B PICTURE 'S 999 V.99'; определяет, что B — вещественная десятичная переменная с фиксированной точкой с точностью (5,2). DCL A PICTURE '$ $, $$9. 99'; определяет вещественное десятичное число с фиксированной точкой с точностью (6,2) и с плавающим знаком доллара. DCL C PICTURE' — 99999E — 99'; показывает вещественное десятичное число с плавающей точкой с точностью (5). При отсутствии символа V считается, что десятичная точка находится непосредственно перед E.

Если после спецификации шаблона стоит описатель COMPLEX, то это означает, что спецификация относится как к действительной, так и к мнимой части комплексного числа.

7.4. СТРУКТУРЫ

В Фортране, а до сих пор и в нашем описании ПЛ/1, данные хранятся в ячейках памяти либо в виде идентификаторов без индексов, либо в виде массивов. Массив представляет собой удобный метод хранения информации, которая должна быть обработана в некотором порядке. Кроме того, ПЛ/1 дает дополнительные возможности выполнения операций и вычисления функций над массивами, что позволяет

программисту обрабатывать этот тип иифорнации с помощью простых и прямых операторов. Осиовиое ограиичение при работе с массивами заключается в том, что каждая ячейка памяти массива должна содержать одии и тот же тип даиных.

Часто возиикает иеобходимость хранить связанную информацию для обработки различных типов данных. Такой порядок требует создаиия отдельиого массива для каждого типа данных, а для установлеиия соответствия между ними введены соответствующие иидексы. Это можио проиллюстрировать на простой задаче по вычислеиию средиего балла для группы студентов. Предположим, что иа карте отперфорирована фамилия каждого студеита вместе с количеством баллов за каждый из 10 курсов. Требуется вычислить средиий балл и рассортировать баллы по убыванию. В выходных даиных должна быть иапечатана фамилия студеита, средиий балл и баллы в порядке убывания.

Предположим, что в группе 100 студентов. Для хранеиия фамилий студентов потребовался бы массив размерности 100; для хранения информации о баллах, полученных студеитами, потребовался бы еще один массив размериости 100 × 10, а для хранения вычислеииого средиего балла — третий массив размерности 100. (Для хранения полученных студентами баллов и их среднего балла можно использовать один массив размерности 100 × 11, так как тип данных тут одииаков.)

Соответствие между фамилиями, полученными баллами и средним баллом определяется индексом каждого массива.

Описание структур. Приведениый пример иа отиосительно простом материале показывает тот тип иифорнации, который в ПЛ/1 может рассматриваться как структура. В Фортраие такого способа описания информации иет, но в Коболе он применяется. *Структура* — это упорядочениая совокупиость даниых различиых типов, которая располагается так, чтобы эту иифорнацию было удобно обрабатывать. Оргаиизация структуры во миогом аиалогичиа оргаиизации сообщения с различными уровиями и подуровиями. Для того чтобы показать архитектуру структуры, используются иомера уровией.

Программист присваивает структурам имена в операторе DECLARE. В структуре могут быть еще и подструктуры. Номера уровней тоже присваиваются программистом. Старшей структуре присваивается иомер 1, а каждой подструктуре присваивается более высокий иомер (ие обязательно последовательиый). Номер уровня структуры появляется перед идеитификатором структуры. Между номером и идентификатором стоит по крайней мере один пробел. Описатели идентификаторов внутри структуры присваиваются по общему правилу — они следуют за имеием идентификатора. Для обработки сложных данных часто употребляются комбинации структур и массивов. Пример, который был приведен в иачале этого параграфа, можио описать как структуру следующим образом:

```
DCL 1 GRADES,
      2 STUDENT_NAME CHAR(10),
      2 COURSE(10) FIXED(3),
      2 AVERAGE FIXED(3);
```

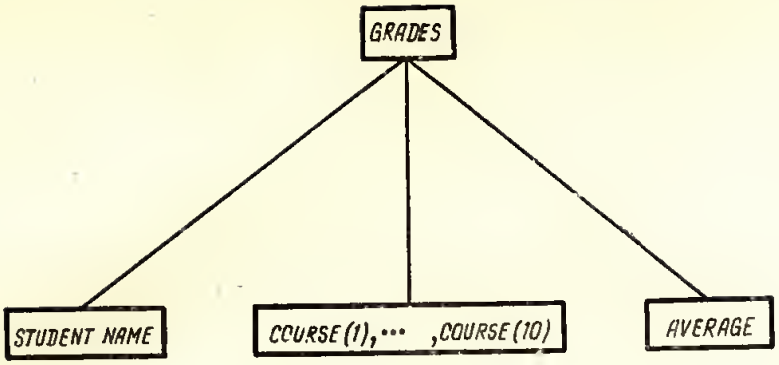


Рис. 7.1.

Организация этой структуры показана на рис. 7.1. При такой спецификации карты могут обрабатываться последовательно по одной. Для облегчения перфорирования повторяющийся номер уровней и описатели можно вынести за скобки. Тогда приведенный ранее оператор можно записать так:

```
DCL 1 GRADES,
      2 (STUDENT_NAME CHAR(40), (COURSE(10), AVERAGE) FIXED(3));
```

Если всю эту информацию необходимо хранить в памяти машины, то нужно сделать следующее описание:

```
DCL 1 GRADES (100),
      2 (STUDENT_NAME CHAR(40), (COURSE(10), AVERAGE) FIXED(3));
```

Этот оператор объявляет GRADES массивом из 100 структур, причем каждая структура состоит из трех идентификаторов уровня 2, а один из этих идентификаторов будет массивом размерности 10. (В подмножестве ПЛ/1 употребление массивов структур запрещено.)

Если же необходимо присвоить различные веса каждому из десяти курсов, записать в разных ячейках имя студента, инициал его второго имени и фамилию, а также хранить всю эту информацию, то нужно сделать следующее объявление:

```
DCL 1 GRADES,
      2 EXAM_WEIGHT (10) FIXED (2),
      2 STUDENT (100),
        3 NAME,
          4 LAST CHAR (15),
          4 FIRST CHAR (15),
          4 MIDDLE CHAR (1),
        3 COURSE (10) FIXED (3),
        3 AVERAGE FIXED (3);
```

Для иллюстрации более сложной структуры рассмотрим такой пример:

```
DCL 1 ACCT_RECORD,
            2 NUMBER CHAR (10),
            2 NAME,
              5 FIRST CHAR (10),
              5 MID_INIT CHAR (1),
```

```
    5 LAST CHAR (10),
2 ADDRESS,
    4 STREET,
      6 NUMBER FIXED (3),
      6 NAME CHAR (15),
    4 CITY,
      6 NAME CHAR (15),
      6 ZIP_CODE FIXED (5),
    4 STATE CHAR (5),
2 BAL FIXED (7, 2);
```

Эту структуру удобно представить так, как показано на рис. 7.2. Она иллюстрирует запись счета одного лица. Если бы программист намеревался хранить счета 100 клиентов, то либо структура ACCT_RECORD могла бы иметь размерность 100, либо каждая из структур второго уровня имела бы размерность 100.

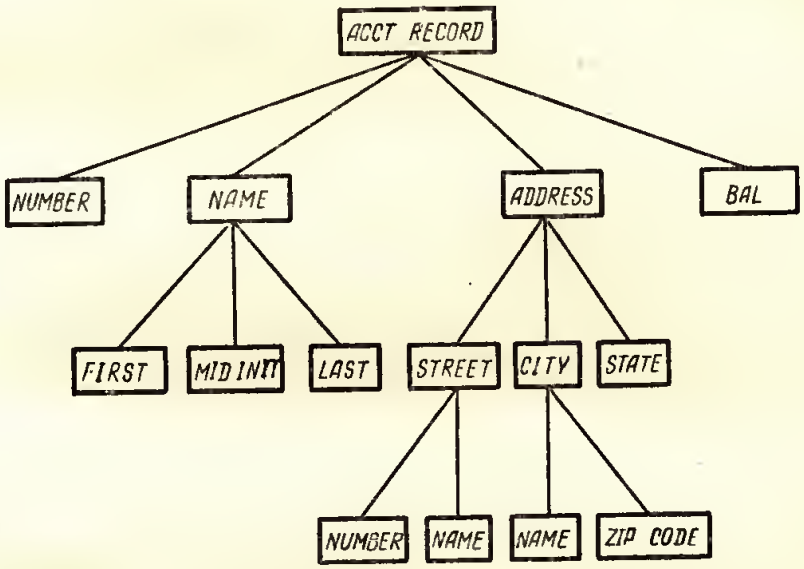


Рис. 7.2.

В приведенном описании используется один и тот же идентификатор на различных уровнях. Выбор имени идентификатора обычно делается для облегчения программирования, но в то же время это может привести к путанице. Любое обращение в программе к идентификатору LAST может вызвать строку символов, представляющую фамилию клиента, в то время как обращение к NAME недостаточно для указания необходимого названия (города, улицы) или имени лица.

Уточняющие имена. Любое имя структуры, за исключением имени старшей структуры, можно уточнить, поставив перед ним имя структуры или подструктуры, к которой данная структура принадлежит. Справа ставится имя с самым большим номером уровня, слева — с самым маленьким номером. Уточнять имя структуры требуется до тех пор, пока оно не станет единственным. В уточняющем имени имена различных уровней отделяются друг от друга точкой, а между точкой и именем может стоять пробел.

Для обращения к названию любой отдельной улицы в структуре ACCT__RECORD можно взять любое из следующих уточняющих имен:

```
ACCT___RECORD.ADDRESS.STREET.NAME
ADDRESS.STREET.NAME
STREET.NAME
```

С помощью уточняющего имени ACCT__RECORD. NAME обращаются к подструктуре, означающей полное имя определенного лица. (В подмножестве ПЛ/1 записи уточняющих имен должны включать имя старшей структуры.)

Для иллюстрации использования индексов в уточняющих именах рассмотрим следующую структуру:

```
DCL 1A (5),
      2 B(10),
        3 E (2, 3),
        3F,
      2 C(8),
        3 E(5),
        3 G(15),
      2 D;
```

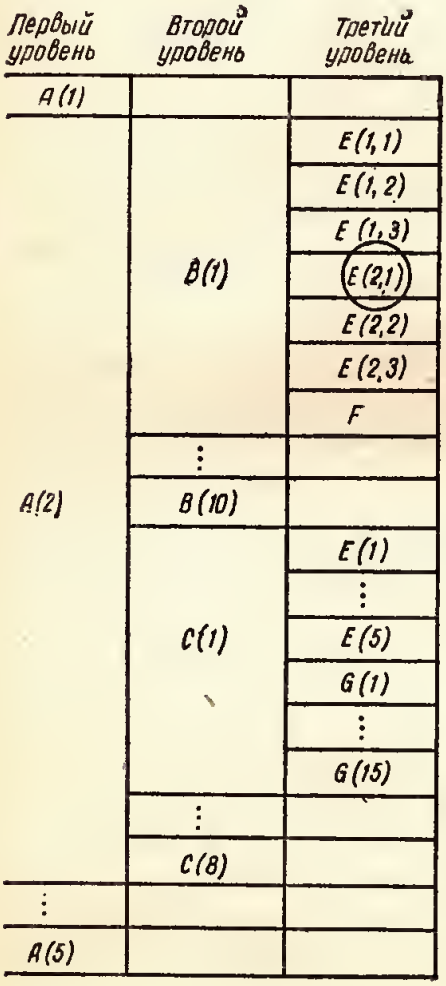


Рис. 7.3,

Схема этой структуры приводится на рис. 7.3.

С помощью уточняющих имен A(2). B (1). E (2,1) можно обращаться к определенному элементу (на рисунке он обведен кружком). Индексы приведенного уточняющего имени могут быть помещены в любое место при условии, что их порядок не будет меняться. Вместо A (2). B (1). E (2,1) можно взять A.B.E. (2,1,2,1), A (2, 1, 2, 1). B. E или A (2). B (1, 2, 1). E (В подмножестве ПЛ/1 индексы ставятся в крайней правой позиции уточняющего имени).

В приведенном примере идентификатор F единственный, он появляется в примере только один раз. Однако для обозначения определенного F требуются два индекса. В этом смысле он является двумерным, зависи как от A, так и от B. Далее перечисляются возможные обращения к одному и тому же элементу: A (1). B (4). F, A (1,4). B.F, A. B. F (1,4), B (1,4). F, B. F (1,4), F (1,4).

Описатель LIKE (Подобен). Описатель LIKE очень удобен для опи-

сания двух одинаковых структур или подструктур. Запись двух приведенных описаний эквивалентна.

```
DCL 1 TAXES,
      2 FEDERAL,
        3 WITH___HELD,
        3 PAID___DIRECT,
      2 STATE,
        3 WITH___HELD,
        3 PAID___DIRECT,
      2 LOCAL,
        3 WITH___HELD,
        3 PAID___DIRECT;
DCL 1 TAXES,
      2 FEDERAL,
        3 WITH___HELD,
        3 PAID___DIRECT,
      2 STATE LIKE FEDERAL,
      2 LOCAL LIKE FEDERAL;
```

Описатель LIKE воспроизводит имена, описатели и размерности, появляющиеся только после указанного идентификатора. Если определяется размерность идентификатора, в описание которого включен описатель LIKE, то она должна быть соответствующим образом обеспечена. Если в первом операторе DECLARE ранее приведенного примера LOCAL имеет размерность 10, то последний оператор второго описания должен быть 2 LOCAL (10) LIKE FEDERAL.

Две структуры или подструктуры, объявленные одинаковыми, необязательно должны быть одного уровня или даже находиться в одной и той же старшей структуре. Если в программе было сделано приведенное выше описание, то при объявлении новой структуры можно включить в запись данную подструктуру (скажем, на уровне 6):

```
6 TAXES PAID LIKE TAXES.
```

Выражения со структурами. Имена структур или подструктур могут использоваться в выражениях примерно так же, как имена массивов. Все структуры, которые появляются в некотором выражении, должны быть одинаково организованы, хотя номера уровней и описатели данных у них могут быть различными. Преобразование данных производится по обычным правилам. Если элементы структур или сами структуры представляют собой массивы, то они должны быть одной размерности.

Рассмотрим следующее объявление:

```
DCL 1 A, 2 AA, 2 AB, 2 AC,
    1 B, 2 BA, 2 BB, 2 BC,
    1 C, 2 CA, 2 CB, 2 CC;
```

Оператор A = B + 2* C; эквивалентен трем операторам:

```
AA=BA+2* CA;
AB=BB+2* CB;
AC=BC+2* CC;
```

Если объявлено, что переменные A, B и C в приведенном примере имеют размерность 15, то оператор A (I) = B (J) + 2* C (K); эквивалентен трем операторам:

```
AA (I)=BA (J)+2* C(K);
AB (I)=BB (J)+2* CB(K);
AC (I)=BC (J)+2* CC (K);
```

где все I, J и K могут принимать любое целое значение от 1 до 15.

Имена структур и имена массивов не могут появляться в одном и том же выражении, однако разрешаются выражения, состоящие из массива структур. Обозначение сечения структуры звездочкой запрещено.

Имя структуры может также стоять в списке ввода-вывода, или в списке фактических параметров аналогично имени массива. При вводе-выводе, управляемом списком или редактированием значения поступают в поток в том порядке, в котором они объявляются оператором DECLARE. Если переменная A описана:

```
DCL I A, 2 B(5), 2 C (10);
```

то оператор PUT LIST (A); отпечатает следующие выходные данные:

```
A. B(I), ... , A. B (5)  A. C (I), ... , A. C (10)
```

Если переменная A описана:

```
DCL I A (5), 2 B, 3 C, 3 D, 2 E
```

то оператор PUT LIST (A); отпечатает следующие данные:

```
A. B. C (I), ... , A. B. C (5), A. B. D (1), ... , A. B. D (5), A. E (I), ..., A. E (5)
```

При вводе-выводе, управляемом данными, имена элементов данных должны быть полностью уточнены, и все индексы должны находиться в крайней правой позиции. При вводе, управляемом данными, имя в списке данных (если оно появляется) не может содержать никаких индексов. Например, если переменная A описана:

```
DCL I A(5), 2 B, 2 C, 3 D, 3 E(10);
```

то при вводе, управляемом данными, входные значения должны иметь вид: A.B (2) = 14.7, A.C.E (5,7) = 10.8 и т. д.

Присваивание по имени BY NAME. (В подмножестве ПЛ/1 присваивание по имени запрещено.) В предыдущем параграфе рассматривались операции над структурами и операторы присваивания. В последнем случае две структуры должны были иметь одинаковое строение. Добавляя слова BY NAME, которые отделяются от выражения запятой, можно производить присваивание и операции над соответствующими именами в выражении, описывающем структуру. В таких выражениях номера уровней не обязательно должны быть одинаковыми.

В качестве иллюстрации рассмотрим следующее объявление:

```
DCL 1 OLD, 2 ODATE, 2 AMOUNT,
    1 NEW, 2 NDATE, 2 AMOUNT, 2 PER__CENT__INCREASE,
    1 TOTAL, 2 AMOUNT 2 DATE, 2 PER__CENT__INCREASE;
```

Оператор TOTAL = OLD + NEW, BY NAME; эквивалентен оператору TOTAL. AMOUNT = OLD. AMOUNT + NEW. AMOUNT; Несмотря на то что PER _ CENT _ INCREASE появляется как в NEW, так и в TOTAL, никакого добавления не делается, так как PER _ CENT _ INCREASE нет в OLD.

7.5. НЕКОТОРЫЕ ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ ПЛ/1, НЕ РАССМАТРИВАЕМЫЕ В НАСТОЯЩЕЙ КНИГЕ

ПЛ/1 — это язык, обладающий разнообразными возможностями. При необходимости он позволяет программисту осуществлять действенный контроль над работой вычислительной машины и программой, полученной в результате трансляции. Основные черты ПЛ/1, а также ряд черт, которые нельзя назвать основными, были рассмотрены в настоящей книге.

В настоящем параграфе мы кратко представим некоторые из особенностей ПЛ/1, которые до сих пор в книге не рассматривались. Цель их рассмотрения — познакомить читателя с их существованием и заинтересовать его в дальнейшем изучении языка ПЛ/1. Более детально эти черты ПЛ/1 рассматриваются в руководствах по работе на конкретных вычислительных системах.

Асинхронные процедуры. Когда обрабатывается программа на Фортране, каждый оператор выполняется один за другим в соответствии с логикой программы. В процессе решения это может привести к тому, что центральный процессор ЭВМ будет простаивать, пока команды ввода и вывода не будут полностью выполнены. Система буферов, встроенная в большинство современных вычислительных машин, позволяет временно задержать передачу информации между оперативной памятью и внешними запоминающими устройствами так, чтобы центральный процессор мог продолжать выполнение программы. Однако и буфер не всегда решает задачу полной загрузки процессора, так как нельзя осуществить передачу большого объема информации со скоростью, требуемой центральным процессором. Программист может получить больший эффект на электронной вычислительной машине при размещении в программе своих команд ввода-вывода.

ПЛ/1 дает возможность асинхронного обращения к процедурам, т. е. после выполнения оператора CALL можно передать управление оператору, следующему непосредственно за оператором CALL. Вызванная процедура считается TASK (задачей) и ей присваивается некоторый приоритет. Всякий раз, когда происходит перерывание, выполняется TASK с самым высоким приоритетом. Может быть создана цепь таких задач с различными приоритетами.

При написании программ с помощью такого метода программист не может быть уверен в фактическом порядке выполнения отдельных частей программы. Одни операторы должны выполняться раньше других, а различные команды ввода обращаются к одному и тому же устройству ввода. Поэтому программист должен проверить, окончена ли данная TASK, или с помощью оператора WAIT ждать до тех пор, пока она не будет закончена. Подобная проверка может быть произведена для того, чтобы выяснить, достигнуто ли в TASK определенное EVENT (событие).

Описатели выравнивания и упаковки. Эти описатели служат только для описания массивов или структур строк данных. Они предоставляют программисту возможность выбора между экономией памяти

и скоростью выполнения программы. Описатель PACKED означает, что между элементами массива или структуры не должно быть неиспользованных областей памяти. Описатель ALIGNED означает, что каждый элемент массива или структуры должен записываться, начиная с определенного места памяти. Это может привести к тому, что между элементами появятся неиспользованные области памяти.

Шаблоны для обозначения стрелинговой денежной системы. В ПЛ/1 предусмотрены символы для редактирования ввода-вывода и вычисления данных, представляющих собой Британскую стерлинговую денежную систему (фунты, шиллинги и пенсы).

Ввод-вывод. До сих пор предполагалось, что во всех случаях ввода и вывода информации устройства ввода-вывода назначаются по умолчанию системой как системные устройства ввода-вывода и что она готова их использовать. Но ПЛ/1 позволяет применять другие устройства ввода-вывода и активизировать или маскировать их во время выполнения программы.

Набор элементов данных на внешнем устройстве называется *файлом*. Эти элементы могут группироваться в несколько файлов, каждый из которых имеет свой собственный идентификатор и описатели. Например, для определения характера передачи данных с именем файла может быть связан один из следующих описателей: INPUT, OUTPUT или UPDATE. Способ, которым должны обрабатываться элементы данных в файле, задается описателями STREAM и RECORD. Перезапись STREAM, как это было рассмотрено ранее, определяет непрерывный поток элементов данных в символьной форме, при перезаписи RECORD предполагается, что файл состоит из групп, содержащих один или несколько элементов данных, представленных в любой форме.

Средства управления трансляцией. Обычно программист не может управлять программой во время трансляции. ПЛ/1 предусматривает две стадии трансляции: предпроцессорную и процессорную. Во время процессорной стадии программа транслируется в выходную рабочую программу. На предпроцессорной стадии исходная программа сканируется и обрабатываются предпроцессорные операторы, процедуры или блоки BEGIN. Большинство операторов ПЛ/1, рассмотренных в настоящей книге, могут быть преобразованы в предпроцессорные путем добавления символа % непосредственно перед оператором. Однако работа с предпроцессорными операторами подчиняется определенным правилам.

Предпроцессорные операторы вставляются в операторы ПЛ/1, которые должны составлять исходную программу. Они могут служить для модификации исходной программы до того, как она будет транслирована в выходную программу. Эти операторы могут вводить в исходную программу ряд операторов ПЛ/1, хранящихся на внешних устройствах. Они могут изменять любой идентификатор программы или определять, что транслироваться должны только определенные части программы. Следующие примеры показывают применение некоторых из этих правил.

В результате предпроцессорной обработки последовательности операторов

```
% DCL X CHAR(9), Y FIXED;
% X='SQRT (Y*Z)';
% Y=15;
A=X-4*C;
```

в текст программы будет вставлен оператор

```
A=SQRT (0015*Z)-4*C;
```

Если создана библиотека сегментов программы и ей присвоено имя LIB, то оператор

```
% INCLUDE LIB (ABC);
```

введет в программу элемент ABC библиотеки LIB.

Для ускорения выполнения цикла DO можно воспользоваться следующей последовательностью операторов:

```
% DCL I FIXED;
% DO I=1 TO 4;
   A(I) =B(I)+3*C(I);
% END;
% DEACTIVATE I;
```

Первый оператор активизирует предпроцессорную переменную I, а последний указывает конец области ее определения, т. е. в последующих операторах она использоваться не может. В результате генерируется последовательность операторов

```
A(1)=B(1)+3*C(1);
A(2)=B(2)+3*C(2);
A(3)=B(3)+3*C(3);
A(4)=B(4)+3*C(4);
```

которая и записывается в предпроцессорную программу. Точность этих индексов определяется по умолчанию. Заметьте, что эта запись не эквивалентна следующему циклу DO, имеющемуся в исходной программе

```
DO I=1 TO 4;
A(I) =B(I)+3*C(I);
END;
```

так как он требует дополнительных вычислений индекса I и проверки его значения для выхода из цикла.

Рассмотрим еще один пример:

```
% DECLARE (READ_IN, PRINT_OUT, PLUS, MINUS, TIMES, DIVIDED_BY, START,
  FINISH) CHARACTER;
% READ_IN='GET LIST';
% PRINT_OUT='PUT SKIP LIST';
% PLUS='+';
% MINUS='-';
% TIMES='*';
% DIVIDED_BY='/';
% START='PROCEDURE OPTIONS (MAIN)';
% FINISH='END';
ABC: START;
```

```
READ_IN (A, B, C, D);
C=(A PLUS B) TIMES C PLUS (A MINUS D DIVIDED_BY C);
PRINT_OUT(C);
FINISH ABC;
```

Текст предпроцессорной программы будет следующим:

```
ABC: PROCEDURE OPTIONS (MAIN);
     GET LIST(A, B, C, D);
     C=(A+B)*C+(A — D/C);
     PUT SKIP LIST (C);
     END ABC;
```

Описанные средства трансляции дают программисту новую возможность улучшения рабочей программы.

Вы ознакомились с основными чертами одного из наиболее совершенных языков программирования высокого уровня ПЛ/1. Может быть, появятся алгоритмические языки ПЛ/2 или ПЛ/п. Но, несмотря на это, авторы надеются, что сведения, полученные из настоящей книги, несомненно, окажутся полезными.

7.6. УПРАЖНЕНИЯ

Короткие упражнения

1. Пусть переменная A описана как индексированный массив размерности (100, 100). Переопределите части массива A как одномерный массив B следующим образом:

а) B соответствует третьей строке массива A;
б) B соответствует двадцать первому столбцу массива A;
в) B соответствует элементам главной диагонали A;
г) B соответствует четным элементам седьмой строки массива A;
д) B соответствует элементам в позициях 1, 2, 4, 8, 16, 32 и 64 главной диагонали массива A.

2. Объясните, как выполняются следующие операторы DECLARE:

```
а) DCL A CHAR(8) INIT ('ABCDEFGH'), B CHAR(2) DEFINED A, C CHAR (3) DEFINED A
   POSITION(4), D CHAR(2) DEFINED C;
б) DCL A (50), Y (50) DEF A (51—1SUB);
в) DCL A (20), B (5, 4) DEFINED A (—4+4* 1SUB+2SUB));
г) DCL A (0 : 9), B (0 : 9) DEFINED A (INDEX(X, 1 SUB));
```

где X содержит строку символов, состоящую из цифр 0, 1, ..., 9. Например, '0135986742'.

```
д) DCL A (10), B (9, 2) DEFINED A (— 1+1SUB+2SUB);
```

3. В следующем примере покажите, как будут редактироваться исходные данные с помощью шаблонов:

Исходные данные	Спецификация шаблонов	Исходные данные	Спецификация шаблонов
а) 123456	ZZZZV.9	и) 123456	$9,999.99
б) 1234	—9999	к) —0012.34	$$,$$9.99DB
в) —1234	—9999	л) 001000	***999
г) 123456	99/99/99	м) 123	99BZ
д) 010203	YY9YYY	н) 1234	ZZV99
е) 000123	Z99999	о) 12.34	ZV999
ж) 000123	ZZZZ99	п) —12.34	SZZV.999
з) 000123	ZZZZZZ$	р) 12.34	—ZZV.99

4. Приведено описание структуры, показывающей запись счета очков игроков в баскетбол:

```
DCL 1 PLAYER,
    2 NAME CHAR (30),
    2 TEAM CHAR (10),
    2 TOTAL_RECORD,
      3 FIELD_GOALS,
        4 ATTEMPTED FIXED (5);
        4 MADE  FIXED (5);
      3 FOULS,
        4 ATTEMPTED FIXED (5),
        4 MADE FIXED (5),
    2 SEASON_RECORD LIKE TOTAL_RECORD,
    2 LAST_GAME_RECORD LIKE TOTAL_RECORD;
```

Ответьте на следующие вопросы:

а) сколько различных элементов представляет каждое из следующих имен?

1) PLAYER, 2) TEAM, 3) TOTAL_RECORD, 4) FOULS, 5) MADE, 6) FOULS. MADE, 7) SEASON_RECORD. FOULS. ATTEMPTED

б) напишите сегменты программы, которые позволяют: 1) хранить в X общее число очков игрока во всех сыгранных играх. За гол в ворота засчитывается два очка, за безрезультативную игру — 1 очко; 2) обновить результаты общего счета игрока и счета сезона, если самая последняя запись счета хранится в подструктуре

LAST_GAME_RECORD

5. Ответьте на вопросы по следующей процедуре:

```
(CONVERSION): A . PROCEDURE;
                    .
                    .
                    .
  (NOUNDERFLOW): IF X/Y   < 3*Y THAN X=Y+Z; ELSE X=Y - Z;
                    .
                    .
                    .
               B: BEGIN;
                    .
                    .
                    .
                  END B;
                    .
                    .
                    .
(NOOVERFLOW) : X=Y*Z/W;
                    .
                    .
                  CALL C;
                    .
                    .
                    .
                END A;
              C: PROCEDURE;
                    .
                    .
                    .
```

```
DO WHILE (X — Y > Z);
   X=Y — Z;
   Y=3*Z;
   END;
     .
     .
     .
END C;
```

а) какова область действия ситуации CONVERSION?

б) каким образом можно маскировать ситуацию CONVERSION для всего блока начала B?

в) как можно избежать прерывания при делении на нуль во всех операторах процедур A и C, но не в B?

г) каково значение приставки (NOUNDERFLOW), которая добавлена к оператору IF в процедуре A?

д) покажите, как вы будете иницировать ситуацию SIZE для всего цикла DO в процедуре C;

е) напишите оператор ON, который будет передавать управление оператору с меткой CON, если во время выполнения процедуры A возникает ситуация CONVERSION.

Задачи для программирования

1) Напишите программу, чтобы помочь преподавателю выставить оценки во всех группах студентов, с которыми он занимался в течение текущего семестра. Для описания переменных используйте структуры. Эта программа должна выполнить за преподавателя всю необходимую в этом случае работу и выдать результаты в таком виде, чтобы преподавателю было легко решить, какую оценку ставить каждому студенту. Программа должна быть по возможности общей, а вход в программу — по возможности свободный, чтобы ею легко было пользоваться. В этой ситуации, вероятно, необходимо сочетание операторов DATA, LIST и EDIT.

При планировании программы должно быть учтено следующее:

а) в программе нужно предусмотреть различное число групп и различное число студентов в каждой группе;

б) количество баллов по данному экзамену не может превосходить 100;

в) преподаватель может пожелать ввести различные веса оценок за различные экзамены;

г) для каждого студента должна быть заведена отдельная карта, так как может потребоваться не только фамилия студента, но и другие данные;

д) программа, вероятно, будет выполняться не один раз для каждой группы и для выставления баллов за определенный период;

е) преподаватель может в любое время пожелать получить список студентов группы без списка баллов;

ж) для каждой группы на каждой странице выходных данных должен быть заголовок и дата;

з) выходные данные должны включать общие сведения о каждом студенте, его баллы, используемые веса и средний балл;

н) было бы желательно иметь два листинга: один, составленный в порядке считывания фамилии студента (для проверочного считывания), другой, составленный в порядке убывания по среднему баллу успеваемости. В последнем случае нужно группировать фамилии студентов по среднему баллу, округленному до десятков: 90, 80 и т. д.;

к) полезной может быть информация о количестве студентов, числе экзаменов, о самом высоком балле, который можно получить на каждом экзамене, о среднеарифметическом и средневзвешенном баллах для каждой группы;

л) в качестве подпрограммы для получения графика выходных данных для каждой группы можно воспользоваться подпрограммой из упражнения 5 главы 5;

м) у некоторых студентов баллы могут быть не выставлены по различным причинам: студент пропустил курс, но данные о нем сохраняются для отчета; студент не сдавал экзамена, но не был от него освобожден (выставляют нулевой балл и это должно найти отражение в выходных данных); студент не сдавал экзамена, потому что был от него освобожден, и сдавать его не будет (в выходных данных это должно быть отражено); студент является вольным слушателем;

н) может возникнуть необходимость изменить некоторые оценки после того, как часть их уже будет отперфорирована. Следует предусмотреть средства для того, чтобы это было легко сделать;

о) нужно отперфорировать карты с комментарием, содержащим инструкции по использованию программы;

п) преподаватель может воспользоваться программой для целой группы, применяя две или три различные системы весов оценок.

2. Данные, представляющие собой счета на долг, отперфорированы на картах следующим образом:

кол. 1 — 10 фамилия
кол. 11 — 20 имя
кол. 21 — инициал второго имени
кол. 26 —30 номер счета
кол. 31 — 45 улица, номер дома
кол. 46 — 55 город
кол. 56 — 57 штат
кол. 58 — 62 ZIP код
кол. 65 — 70 баланс на время последнего взноса
кол. 75 — 80 сумма, которую необходимо выплатить со времени последнего взноса

Вся символьная информация, отперфорированная в поле данных, левоустановленная, а вся числовая информация правоустановленная. Данные о денежных расчетах в колонках 65—80 выражены в долларах и центах и отперфорированы без десятичной точки.

Считайте эти карты и отпечатайте ежемесячный отчет о каждом счете. В отчете должны быть отпечатаны фамилия клиента, адрес, номер счета, последний итог, сделанные взносы (если они были сделаны), пеня, новый итог и сумма, которую необходимо выплатить. Новый итог равен последнему итогу — сделанные взносы + пеня (пеня равна 1,5% суммы последнего итога). Требуемая для выплаты сумма равна 10% нового итога или 5 долларам (в зависимости от того, какая сумма больше). Если внесимый взнос не покрывает требуемой суммы, то налагается штраф из расчета 2%.

Для редактирования ввода-вывода воспользуйтесь структурами и шаблонами. Организуйте упорядоченный вывод данных, отпечатайте каждый индивидуальный счет и включите сумму, показывающую общий итог.

ПРИЛОЖЕНИЕ А. СИМВОЛЫ И КОМБИНАЦИИ СИМВОЛОВ НА ПЕРФОКАРТАХ И В КОДАХ EBCDIC

60-символьный алфавит				48-символьный алфавит		
символы и комбинации символов		перфокарты	8-битовый код	символы и комбинации символов	перфокарты	8-битовый код
Буквенные						
Знак доллара	$	11—8—3	0101 1011	$ (см. примеч. 3)	11—8—3	0101 1011
Знак номера	#	8—3	0111 1011			
	A	12—1	1100 0001	A	12—1	1100 0001
	B	12—2	1100 0010	B	12—2	1100 0010
	C	12—3	1100 0011	C	12—3	1100 0011
	D	12—4	1100 0100	D	12—4	1100 0100
	E	12—5	1100 0101	E	12—5	1100 0101
	F	12—6	1100 0110	F	12—6	1100 0110
	G	12—7	1100 0111	G	12—7	1100 0111
	H	12—8	1100 1000	H	12—8	1100 1000
	I	12—9	1100 1001	I	12—9	1100 1001
	J	11—1	1101 0001	J	11—1	1101 0001
	K	11—2	1101 0010	K	11—2	1101 0010
	L	11—3	1101 0011	L	11—3	1101 0011
	M	11—4	1101 0100	M	11—4	1101 0100
	N	11—5	1101 0101	N	11—5	1101 0101
	O	11—6	1101 0110	O	11—6	1101 0110
	P	11—7	1101 0111	P	11—7	1101 0111
	Q	11—8	1101 1000	Q	11—8	1101 1000
	R	11—9	1101 1001	R	11—9	1101 1001
	S	0—2	1110 0010	S	0—2	1110 0010
	T	0—3	1110 0011	T	0—3	1110 0011
	U	0—4	1110 0100	U	0—4	1110 0100
	V	0—5	1110 0101	V	0—5	1110 0101
	W	0—6	1110 0110	W	0—6	1110 0110
	X	0—7	1110 0111	X	0—7	1110 0111
	Y	0—8	1110 1000	Y	0—8	1110 1000
	Z	0—9	1110 1001	Z	0—9	1110 1001
Числовые	0	0	1111 0000	0	0	1111 0000
	1	1	1111 0001	1	1	1111 0001
	2	2	1111 0010	2	2	1111 0010
	3	3	1111 0011	3	3	1111 0011
	4	4	1111 0100	4	4	1111 0100
	5	5	1111 0101	5	5	1111 0101
	6	6	1111 0110	6	6	1111 0110
	7	7	1111 0111	7	7	1111 0111
	8	8	1111 1000	8	8	1111 1000
	9	9	1111 1001	9	9	1111 1001
Специальные символы и комбинации символов						
Пробел		не перфорируется	0100 0000		не перфорируется	0100 0000
Точка	.	12—8—3	0100 1011	.	12—8—3	0100 1011
Знак «меньше»	<	12—8—4	0100 1100	LT	11—3,0—3	

60-символьный алфавит			48-символьный алфавит		
символы и комбинации символов	перфокарты	8-битовый код	символы и комбинации символов	перфокарты	8-битовый код
Открывающая скобка (	12—8—5	0100 1101	(	12—8—5	0100 1101
Плюс +	12—8—6	0100 1110	+	12—8—6	0100 1110
Символ «или» |	12—8—7	0100 1111	OR	11—6,11—9	
Символ «и» &	12	0101 0000	AND	12—1, 11—5,12—4	
Звездочка *	11—8—4	0101 1100	*	11—8—4	0101 1100
Закрывающая скобка)	11—8—5	0101 1101	)	11—8—5	0101 1101
Точка с запятой ;	11—8—6	0101 1110	;.	0—8—3, 12—8—3 (см. примеч. 6)	
Символ «нет» ¬	11—8—7	0101 1111	NOT	11—5,11—6,0—3	
Минус —	11	0101 0000	—	11	0110 0000
Наклонная черта /	0—1	0101 0001	/	0—1	0110 0001
Запятая ,	0—8—3	0110 1011	,	0—8—3	0110 1011
Процент %	0—8—4	0110 1100	//	0—1, 0—1 (см. примеч. 5)	
Символ разбивки _	0—8—5	0110 1101		(см. примеч. 3)	
Знак «больше» >	0—8—6	0110 1110	GT	12—7, 0—3	
Вопросительный знак ? (см. примеч. 8)	0—8—7	0110 1111		(см. примеч. 3)	
Двоеточие :	8—2	0111 1010	..	(см. примеч. 7) 12—8—3,12—8—3	
Апостроф (кавычка) '	8—5	0111 1101	'	8—5	0111 1101
Знак равенства или присваивания значения :=	8—6	0111 1110	:=	8—6	0111 1110
Меньше или равно <=	12—8—4,8—6		LE	11—3,12—5	
Конкатенация ||	12—8—7, 12—8—7		CAT	12—3,12—1,0—3	
Возведение в степень **	11—8—4, 11—8—4		**	11—8—4,11—8—4	
Не меньше ¬<	11—8—7, 12—8—4		NL	11—5,11—3	
Не больше ¬>	11—8—7,0—8—6		NG	11—5,12—7	
Не равно ¬=	11—8—7, 8—6		NE	11—5,12—5	
Больше или равно >=	0—8—6,8—6		GE	12—7,12—5	
Начало примечания /*	0—1,11—8—4		/*	0—1,11—8—4	
Конец примечания */	11—8—4,0—1		*/	11—8—4,0—1	
Знак указателя ->	11,0—8—6		PT	11—7,0—3	

Примечания

1. В комбинациях символов пробелы запрещены.

2. При 48-символьном алфавите слова GT, GE, NE, LE, LT, NOT, OR, AND и CAT резервируются и не могут служить идентификаторами.

3. В 48-символьном алфавите символы #, @, ¬ и ?, разрешенные в 60-символьном алфавите, не используются.

4. Для отделения символа от идентификаторов в случае комбинации символов необходимо оставить пробелы перед и после комбинации. Например, X GT Y нельзя записать как XGTY.

5. Перед двумя наклонными чертами (или после них), обозначающими знак процента, необходимо оставить пробел, если перед ними (или после них) стоит звездочка.

6. Запятая и точка (,.) могут обозначать знак «;», но если они встречаются в строке символов или в комментарии, или же за ними сразу следует цифра, то они не означают знак «точка с запятой».

7. Если после символа «точка» следуют две точки, означающие двоеточие, то после точки ставится пробел.

8. В настоящее время в ПЛ/1 вопросительный знак не используется.

ПРИЛОЖЕНИЕ Б. ТАБЛИЦА ОПИСАТЕЛЕЙ, ВСТРЕЧАЮЩИХСЯ В КНИГЕ

Описатель	Сокращения[1]	Употребление в подмножестве	Параграфы, в которых они встречались
ALIGNED		да	7.5
AUTOMATIC	AUTO	да	6.4
BASED		да	6.4
BINARY	BIN	да	2.4,3.4,3.5
BIT (длина)		да	3.5
BUILTIN		да	6.5
CHARACTER (длина)	CHAR (длина)	да	3.5
COMPLEX	CPLX	нет	2.4,3.4,3.5
CONTROLLED	CTL	нет	6.4
DECIMAL	DEC	да	2.4,3.4,3.5
DEFINED	DEF	да	7.1
ENTRY		да	6.5
EXTERNAL	EXT	да	6.3
FILE		да	7.5
FIXED		да	2.4,3.4,3.5
FLOAT		да	2.4,3.4,3.5
INITIAL	INIT	нет	3.5,6.4,6.5
INTERNAL	INT	да	6.3,6.4
LABEL		да	3.5
LIKE		нет	7.4
PICTURE	PIC	да	7.3
POSITION (l)	POS (l)	нет	7.1
REAL		нет	2.4,3.4,3.5
RETURNS		да	6.5
STATIC		да	6.4,6.7
UNALIGNED	UNAL	нет	7.5
VARYING	VAR	нет	3.5

[1] В подмножестве ПЛ/1 сокращения запрещены.

ПРИЛОЖЕНИЕ В. ТАБЛИЦА КЛЮЧЕВЫХ СЛОВ (ЗА ИСКЛЮЧЕНИЕМ ОПИСАТЕЛЕЙ И ФУНКЦИЙ), КОТОРЫЕ ВСТРЕЧАЮТСЯ В КНИГЕ

Ключевое слово	Сокращение[1]	Применения	Употребление в подмножестве	Параграфы, в которых они встречаются
ALLOCATE		оператор	нет	6.4—6.6
BEGIN		оператор	да	2.3,3.2, глава 6
BY		конструкция оператора DO	да	2.6,4.7
BY NAME		необязательное слово оператора присваивания	нет	7.4
CALL		оператор или необязательное слово в INITIAL	да	глава 6
CHECK		условие	нет	2.7
COLUMN (w)	COL (w)	элемент формата	да	5.2—5.4

[1] В подмножестве ПЛ/1 сокращения запрещены.

Ключевое слово	Сокращение	Применения	Употребление в подмножестве	Параграфы, в которых они встречаются
CONDITION		условие	нет	7.2, приложение Г
CONVERSION	CONV	условие	да	7.2, приложение Г
COPY		необязательное слово в операторе	нет	2.7,5.2—5.4
DATA		ввод-вывод непрерывным потоком	нет	5.3,5.6
DECLARE	DCL	оператор	да	2.4,3.5,6.3
DISPLAY		оператор	да	5.7
DO		оператор	да	2.6,4.7
EDIT		ввод-вывод непрерывным потоком	да	5.4—5.6
ELSE		конструкция оператора IF	да	2.6,4.8
END		оператор	да	2.3,2.6,3.2, 4.7,6.2
ENDFILE (имя файла)		условие	да	2.7, приложение Г
ENDPAGE (имя файла)		условие	да	7.2, приложение Г
ENTRY		описатель или оператор	да	6.2 (оператор)
ERROR		условие	да	7.2, приложение Г
EXIT		оператор	нет	6.2
FINISH		условие	нет	7.2, приложение Г
FIXEDOVERFLOW	FOFL	условие	да	7.2, приложение Г
FORMAT (список)		оператор	да	5.4
FREE		оператор	иет	6.4
GET		оператор	да	2.7,5.2—5.6
GO TO	GOTO	оператор	да	2.6,4.6,6.2
IF		оператор	да	2.6,4.8
LINE (w)		элемент формата, необязательное слово оператора PUT	да	2.7,5.2—5.4

Ключевое слово	Сокращение	Применения	Употребление в подмножестве	Параграфы, в которых они встречаются
LIST		ввод-вывод непрерывным потоком	да	2.7, 5.2, 5.6
MAIN		необязательное слово оператора PROCEDURE	да	3.2, 6.1, 6.2
NOCHECK		префикс условия	нет	7.2, приложение Г
NOCONVERSION		префикс условия	да	7.2, приложение Г
NOFIXEDOVERFLOW	NOFOFL	префикс условия	да	7.2, приложение Г
NOOVERFLOW	NOOFL	префикс условия	да	7.2, приложение Г
NOSIZE		префикс условия	да	7.2, приложение Г
NOSTRINGRANGE	NOSTRG	префикс условия	нет	7.2, приложение Г
NOSUBSCRIPTRANGE	NOSUBRG	префикс условия	нет	7.2, приложение Г
NOUNDERFLOW	NOUFL	префикс условия	да	7.2, приложение Г
NOZERODIVIDE	NOZDIV	префикс условия	да	7.2, приложение Г
ON		оператор	да	7.2. приложение Г
OPTIONS (список)		необязательное слово в операторе PROCEDURE	да	2.3, 3.2, глава 6
OVERFLOW	OFL	условие	да	7.2, приложение Г
PAGE		элемент формата, необязательное слово в операторе PUT	да	2.7, 5.2—5.4
PROCEDURE	PROC	оператор	да	2.3, 3.2, глава 6
PUT		оператор	да	2.7, 5.2—5.6
RECURSIVE		необязательное слово в операторе PROCEDURE	нет	6.7
REPLY (id)		необязательное слово в операторе DISPLAY	да	5.7
RETURN		оператор	да	6.2
REVERT		оператор	да	7.2, приложение Г
SIGNAL		оператор	да	7,2, приложение Г

Ключевое слово	Сокращение	Применения	Употребление в в подмножестве	Параграфы, в которых они встречаются
SIZE		условие	да	7.2, приложение Г
SKIP [(x)]		элемент формата, необязательное слово в операторах GET или PUT	да	2.7, 5.2—5.4
SNAP		необязательное слово в операторе ON	нет	7.2, приложение Г
STOP		оператор	да	6.2
STRINGRANGE	STRG	условие	нет	7.2, приложение Г
STRING (id)		необязательное слово в операторах GET и PUT	да	5.6
iSUB		фиктивная переменная описателя DEFINED	нет	7.1
SUBSCRIPTRANGE	SUBRG	условие	нет	7.2, приложение Г
SYSIN		имя входного файла стандартной системы IBM	нет	2.7
SYSPRINT		имя выходного файла стандартной системы IBM	нет	2.7
SYSTEM		необязательное слово в операторе ON	да	7.2, приложение Г
THEN		конструкция оператора IF	да	2.6, 4.8
TO		конструкция оператора DO	да	2.6, 4.7
TRANSMIT		условие	да	7.2, приложение Г
UNDERFLOW	UFL	условие	да	7.2, приложение Г
WHILE		конструкция оператора DO	да	2.6, 4.7
ZERODIVIDE	ZDIV	условие	да	7.2, приложение Г

Имя ситуации (сокращение)	Реакция системы по умолчанию	Контроль программы	Употребление в помножестве	Условие возникновения	Стандартная реакция системы
1	2	3	4	5	6
Ситуации вычислений					
CONVERSION (CONV)	разрешена	да	да	В строке символов встречаются данные, не подлежащие преобразованию	Результат не определен, сообщение об ошибке, возникновение ситуации ERROR
FIXEDOVERFLOW (FOFL)	разрешена	да	да	Число с фиксированной запятой превышает допустимое значение	Число усекается слева, сообщение об ошибке, продолжение счета
OVERFLOW (OFL)	разрешена	да	да	Абсолютное значение порядка числа с плавающей точкой превышает допустимое значение	Возникновение ситуации ERROR, сообщение об ошибке, результат не определен
SIZE	запрещена	да	да	Поле результата недостаточно для записи всего результата	Результат не определен, посылка сообщения и возникновение ситуации ERROR
UNDERFLOW (UFL)	разрешена	да	да	Порядок числа с плавающей точкой меньше допустимого минимума	Результат равен нулю, сообщение об ошибке и продолжение счета
ZERODIVIDE (ZDIV)	разрешена	да	да	Попытка разделить на нуль	Результат не определен. Сообщение об ошибке и возникновение ситуации ERROR
Ситуации ввода-вывода (только в операторах ON, SIGNAL и REVERT)					
ENDFILE (имя файла)	разрешена	нет	да	Конец файла	Сообщение об ошибке и возникновение ситуации ERROR
ENDPAGE (имя файла)	разрешена	нет	да	Превышен размер текущей страницы	Печать начинается на новой странице
NAME (имя файла)	разрешена	нет	нет	Попыткз прочесть неправильное имя данных	Сообщение об ошибке и игнорирование вводимого поля
TRANSMIT (имя файла)	разрешена	нет	да	Наличие устойчивой ошибки при передаче данных ввода-вывода	Сообщение об ошибке и возникновение ситуации ERROR

Имя ситуации (сокращение)	Реакция системы по умолчанию	Контроль программы	Употребление в подмножестве	Условие возникновения	Стандартная реакция системы
1	2	3	4	5	6
Ситуации контроля программы					
CHECK (список элементов)	разрешена	да	нет	Появление в списке элементов, вызывающих прерывание	Печать элемента данных и его значение на системном выводе, выполненне программы продолжается
SUBSCRIPTRANGE (SUBRG)	запрещена	да	нет	Значение индекса находится вне заданных границ	Сообщение об ошибке и возбуждение ситуации ERROR
STRINGRANGE (STRG)	запрещена	да	нет	При использовании SUBSTR допущена ошибка в длине списка фактических параметров	Выполненне программы продолжается, длина списка преобразуется в допустимые границы
Ситуации реакции системы					
ERROR	разрешена	нет	да	Нестандартное окончание программы или возникновение сообщения SIGNAL ERROR	Возникновение ситуации FINISH (в подмножестве выполнение программы заканчивается)
FINISH	разрешена	нет	нет	Наличие ошибки, операторы STOP, RETURN, EXIT, END в главной процедуре или сообщение SIGNAL FINISH	Выполненне программы заканчивается
Ситуация, задаваемая программистом					
CONDITION (идентификатор)	разрешена	нет	нет	Возбуждается оператором SIGNAL	Сообщение об ошибке, продолженне выполнения программы

Примечание. В подмножестве ПЛ/1 сокращения запрещены.

Математические функции I: аргументы могут быть вещественными или комплексными (при условии, что они не объявлены иначе); аргументы могут быть массивами; в подмножестве ПЛ/1 комплексные аргументы запрещены; p и q — десятичные целые константы, причем q со знаком.

Имя	Аргументы	Сокращение (не употребляется в подмножестве)	Значение
1	2	3	4
ABS(x)		BIN	абсолютное значение x
ADD (x, y, p [,q])[1]			$x+y$ с точностью (p) в случае с плавающей точкой; (p, q) в случае с фиксированной точкой
BINARY (x[p[,q]])			x преобразуется в двоичное число с точностью (p) в случае с плавающей точкой; (p, q) в случае с фиксированной точкой; если p и q опущены, значению присваивается точность x
CIEL(x)	x не может быть комплексным числом		наименьшее целое число
COMPLEX(x, y)[1,2]	x и y должны быть вещественными числами	CPLX	$x+yi$
CONJ(x)	x должен быть комплексным числом		число, сопряженное x
DECIMAL (x[,p[q]])		DEC	то же, что для BINARY, если только x не преобразован в десятичное число
DEVIDE(x, y, p[,q])[1]			то же, что для ADD, за исключением x/y
FIXED(x[,p[,q]])			x преобразуется в число с фиксированной точкой; если p и q опущены, то применяется система по умолчанию
FLOAT(x[,p])	x не может быть комплексным числом		x преобразуется в число с плавающей точкой; если p опущено, то действует система по умолчанию
FLOOR(x)			наибольшее число $\leqslant x$

[1] В подмножестве ПЛ/1 не допускается
[2] Может служить в качестве псевдопеременной.

Имя	Аргументы	Сокращение (не употребляется в подмножестве)	Значение
1	2	3	4
IMAG(x)[1,2]	x должен быть комплексным числом		мнимая часть x
MAX(x_1, . . ., x_n)	необходимы два или более фактических параметра; не может быть комплексным числом		наибольший из всех фактических параметров
MIN(x_1, . . ., x_n)	то же, что и для MAX		наименьший из всех фактических параметров
MOD(x, y)	x и y не могут быть комплексными числами		положительный остаток от деления x/y
MULTIPLY(x, y, p[,q])[1]		PREC	то же, что и для ADD, за исключением $x*y$
PRECISION(x, p[,q])			x преобразуется в число с точностью (p) в случае плавающей точки и (p, q) в случае фиксированной точки
REAL(x)[1,2]	x должен быть комплексным числом		вещественная часть x
ROUND(x,n)	n десятичная целая константа; может иметь знак		
	Случай 1: x — число с плавающей точкой		второй фактический параметр игнорируется и самый правый бит внутреннего представления устанавливается равным 1
	Случай 2: x — число с фиксированной точкой		округляет x до n-й цифры справа от двоичной или десятичной точки, если $n>0$, до n-й цифры слева, если $n<0$
	Случай 3: x — строка символов		значение x не меняется
SIGN(x)	x не может быть комплексным числом		значение — двоичное число 1, 0, или —1 соответственно при $x>0$, $x=0$, $x<0$
TRUNC(x)	x не может быть комплексным числом		если $x<0$, то значение равно CIEL (x)
			если $x\geqslant 0$, то значение равно FLOOR (x)

[1] В подмножестве ПЛ/1 не допускается.
[2] Может служить в качестве псевдопеременной.

Математические функции II: все аргументы и результаты представляют собой числа с плавающей точкой и при необходимости преобразуются; аргументы могут быть массивами и вещественными или комплексными числами, если только они не объявлены иначе; в подмножестве ПЛ/1 употребление комплексных аргументов запрещено.

Имя	Аргумент	Результаты	Ситуация ERROR
ATAN (x)	вещественный комплексный	$-\pi/2<$арктангенс (x)$<\pi/2$ в радианах $-i^*$ арктангенс (i^*x)	$x=\pm 1i$
ATAN (x, y)	x и y должны быть вещественными	если $y>0$, то результат равен ATAN (x/y) радиан; если $x>0$ и $y=0$, то результат равен $-\pi/2$ раднан; если $x\geqslant 0$ и $y<0$, то результат равен ($\pi+$ATAN(x/y)) радиан; если $x<0$ и $y=0$, то результат $-\pi/2$ радиан; если $x<0$ и $y<0$, то результат равен ($-\pi+$ATAN (x/y)) радиан	$x=y=0$
ATAND (x)	x должен быть вещественным	$-90<$arctan (x)<90 в градусах	
ATAND (x, y)	x и y должны быть вещественными	((180/π)* ATAN (x, y)) в градусах	$x=y=0$
ATANH (x)	x — вещественное число x — комплексное число	arctanh (x) (log ((1+x)/(1−x)))/2	$\lvert x\rvert\geqslant 1$ $\lvert x\rvert=1$
COS (x)	x — вещественное число в радианах x — комплексное число в радианах	cos x cos (Re (x))* cosh (Im (x)) − i* sin (Re (x))* sinh (Im (x))	

Продолжение

Имя	Аргумент	Результаты	Ситуация ERROR
COSD (x)	x — вещественное число в градусах	cos (x)	
COSH (x)	x — вещественное число	cosh (x)	
	x — комплексное число	cosh ($Re\,(x)$)* cos ($Im\,(x)$) + i* sinh ($Re\,(x)$)* sin($Im\,(x)$)	
ERF (x)	x — вещественное число	$\mathrm{ERF}\,(x)=\frac{2}{\pi}\int_0^x e^{-t^2}\mathrm{d}t$	
ERFC (x)	x — вещественное число	1 — ERF (x)	
LOG (x)	x — вещественное число	e(основание натуральных логарифмов) в степени x натуральный логарифм x	$x\leqslant 0$
	x — комплексное число	LOG(ABS (x)) + iv, где $-\pi<v<\pi$	$x=0$
LOG 10 (x)	x — вещественное число	десятичный логарифм x	$x\leqslant 0$
LOG 2 (x)	x — вещественное число	двоичный логарифм x	$x\leqslant 0$
SIN (x)	x — вещественное число в радианах	sin (x)	
	x — комплексное число в радианах	sin ($Re\,(x)$) cosh ($Im\,(x)$) + i* cos ($Re\,(x)$)* sinh ($Im\,(x)$)	
SIND (x)	x — вещественное число в градусах	sin (x)	
SINH (x)	x — вещественное число	sinh (x)	
	x — комплексное число	sinh ($Re\,(x)$)* cos ($Im\,(x)$) + i* cosh ($Re\,(x)$)* sin ($Im(x)$)	$x<0$
SQRT (x)	x — вещественное число	положительный квадратный корень из x	
	x — комплексное число	$u\pm i^*v$, где либо $u>0$, либо $u=0$ и $v\geqslant 0$	
TAN (x)	x — в радианах	tan (x)	
TAND (x)	x — в градусах	tan (x)	
TANH (x)		tanh (x)	

Встроенные функцни строк снмволов: все аргументы, еслн только они ие объявлеиы иначе, представляют собой строки символов или массивы, состоящие нз строк символов. Если функцня содержит более одного аргумента, то к более короткому добавляются нулн, чтобы числю снмволов всех аргументов было одннаково. Те аргументы, которые полагаются строками снмволов, а в действнтельностн таковыми ие являются, должны быть преобразованы в строки битов, еслн они представляют собой двончные числа, нлн в строки символов, если онн десятичные числа.

Имя	Аргументы	Значение результата
BIT (*x* [, *y*])	*x* — выражение, которое должио быть преобразовано *y* — десятичная целая константа	*x* преобразовывается в строку битов длнной *y*; если перемеиная *y* не указана, то длина результата определяется по *x*
BOOL (*x*, *y*, *z*)	(см. параграф 4.4)	
CHAR (*x* [, *y*])	*x* — выражение, которое должио быть преобразовано *y* — десятнчиая целая константа	*x* преобразовывается в строку снмволов длнной *y*; если переменная *y* не указана, то длнна результата определяется по *x*
HIGH (*x*)	десятичная целая коистанта	строка снмволов длиной *x*, причем каждый символ имеет нанбольшее значенне в последовательиости
INDEX (*x*, *y*)		номер самого левого элемента *x*: начиная с него *y* входнт в *x* в качестве подстроки. Еслн *y* не прннадлежнт *x* илн если строка имеет нулевую длину, то результатом будет значенне ОВ
LENGTH (*x*)[1]		двоичное целое число, представляющее текущую длину *x*
LOW (*x*)	десятичная целая константа	строка символов длиной *x*, причем каждый символ имеет иаименьшее значеиие в последовательности
REPEAT (*x*, *y*)	*x* — строка *y* — десятичиая целая константа	*x* конкатеинрует сам с собой *y* раз если $y \leqslant 0$, то результатом будет строка *x*
STRING (*x*)[1]	*x* — имя элемеита, массива или структуры	коикатенация всех элементов в *x*
SUBSTR (*x*, *y* [, *z*])[2]	(см. параграф 4.4)	
UNSPEC (*x*)[2]	*x* ие может быть массивом	внутреиисе представление *x* в форме строки бнтов

[1] В подмножестве ПЛ/1 не употребляется.
[2] Может служить как псевдопеременная.

Фуикции массивов: все аргументы представляют собой массив, результатом являются зиачения элемеитов массива, если только они не объявлены иначе.

Имя	Аргументы	Значенне результата
ALL (*x*)	массив из строк битов	строка битов, где *i*-й бит равен 1, если в *x* имеется *i*-й бит каждого элемента и ои равеи 1; в противном случае *i*-й бит равен 0
ANY (*x*)	массив из строк битов	строка битов, где *i*-й бит равен 1, если хоть в каком-нибудь из элементов *x* имеется *i*-й бит и он равен 1; в противиом случае *i*-й бит равеи 0
DIM (*x*, *n*)[1]	*n* — двоичное целое число	двоичиое целое чнсло, задающее текущую *n*-ю размерность *x*
HBOUND (*x*, *n*)[1]	*n* — двоичное целое число	двоичное целое число, задающее текущий верхний предел *n*-й размериости *x*
LBOUND (*x*, *n*)[1]	*n* — двоичное целое число	двоичное целое чнсло, задающее текущнй нижиий предел *n*-й размерности *x*
POLY (*x*, *y*)[1]	*x*, *y* — одиомерные массивы	многочлен от *x* н *y*
PROD (*x*)		произведение всех элемеитов в *x*
SUM (*x*)		сумма всех элементов в *x*

[1] В подмножестае ПЛ/1 не употребляется.

Различные функцнн

Имя	Аргументы	Значенне результата
ALLOCATION (*x*)[1]	имя элемента, массива или старшей структуры с описателем CONTROLLED	'1' В, если память выделяется для X, определеиного в иастоящем блоке; в противиом случае — '0' В,
COUNT(имя файла)[1]	должеи иметь описатель STREAM	двоичиое целое число с фиксированной точкой, задающее число элемеитов, передаваемых в последней операции ввода-вывода по имени файла
DATE	нет	строка символов 'yymmdd', где yy — текущий год, mm — текущий месяц, dd — текущнй деиь
LINENO(имя файла)[1]	должен нметь описатель PRINT	двоичиое целое число с фнксированной точкой, задающее текущий номер строки имени файла
TIME	нет	строка символов 'hhmmsstt', представляющая текущее время, где hh — часы, mm — минуты, ss — секунды, tt — миллисекуиды

[1] В подмножестве ПЛ/1 не употребляется.

Фуикции-ситуации: аргумеитов не содержат, могут употребляться только в ON-units или блоках, активизированиых ON-units

Имя	Зиачение результатов
DATAFIELD[1]	строка символов VARYING, содержащая поле даниых, вызвавших последиее прерывание NAME
ONCHAR[1,2]	строка символов длиной I, содержащая символ, который вызвал последнее прерывание CONVERSION
ONCODE[1]	двоичное целое число, определяющее своим значением последнее прерываиие
ONCOUNT[1]	двоичное зиачеиие, дающее число прерываний, включая последиее
ONFILE[1]	строка символов VARYING, содержащая имя файла, которое вызвало прерывание при вводе-выводе или преобразоваиии
ONLOC[1]	строка символов VARYING, содержащая имя точки входа в процедуру, которая вызвала прерывание
ONSOURCE[1,2]	строка символов VARYING, описывающая имя поля, которое вызвало прерываиие при ситуации CONVERSION

[1] В подмножестве ПЛ/1 не употребляется.
[2] Может служить в качестве псевдопеременной.

ОТВЕТЫ К КОРОТКИМ УПРАЖНЕНИЯМ

Глава 2

1. Примеры в), г) и д) правильны; а) начинается не с алфавитного символа; б) содержит слишком много символов.

2. Примеры б), г), д) правильны; а) содержит два символа, которые не являются буквенно-цифровыми символами или разделителями, в) содержит пробел, который не является буквенно-цифровым символом или разделителем.

3. Пример а) правилен, б) неправилен; нет точки с запятой; в) правилен (START — это метка, а не ключевое слово); г) неправилен, символ операции не может стоять слева от символа '='; д) правилен; е) неправилен; 4A неверное имя переменной, оно не может означать 4 × A; ж) неправилен; X, Y, Z необходимо заключить в круглые скобки; з) неправилен; CHECK (SUM, X, Y) нужно заключить в круглые скобки так же, как и MA1N; и) правилен; к) неправилен; $10.00 не может быть меткой имени, так как в нем содержится специальный символ (.); л) правилен.

4. Обратите внимание на то, что запись 1FA*B > 0 THEN X = 5; X = 4; неправильна, так как тогда оператор X = 4 всегда будет выполняться. Один из возможных ответов:

```
IF A*B > 0  THEN X=5;
            THEN X=4;
```

5. а) A = 45, B = 45.00, C = . 1234500E5; б) A = 00, B = = 00.12, C = . 1234500E0.

6. а) A=1111B, B=0111.11B б) 15 7.75 в) A=0100B, B=0100.00B (заметьте, что 20 (основание 10) = 10100 (основание 2)); г) 4 4.00.

7. а) В результате будет 9 значений Z. Число напечатанных строк зависит от того, сколько цифр в каждой строке печатает та или иная система; б) 396 строк; в) 26 строк.

8. а) ((A**B)*2) + (3.456* Y)

б) (A + (B*C)) — (D/F)

в) (A = B) | ((A < C) & (—|(B—| < (C**2))))

г) ((A**(B**C)) — D) = ((A*B)/C)

9. а) (A + B)/C

б) A + B/C

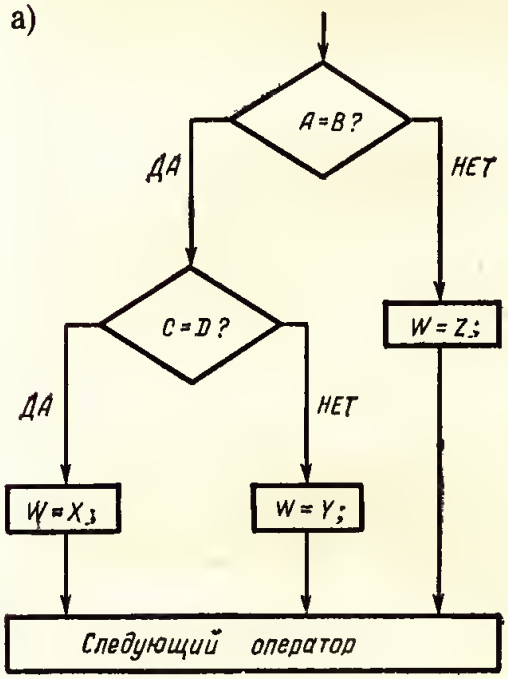
в) $\neg(\neg(A < B) | C >= D)$

г) A + B — (C + D) + 2*E

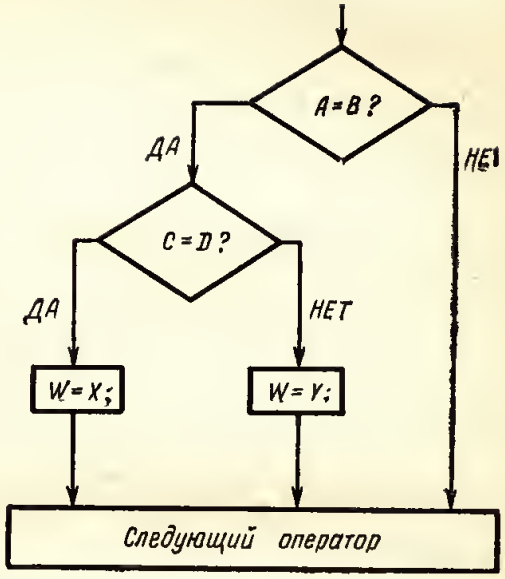
д) A + B + (C + D)* 2*E

10.

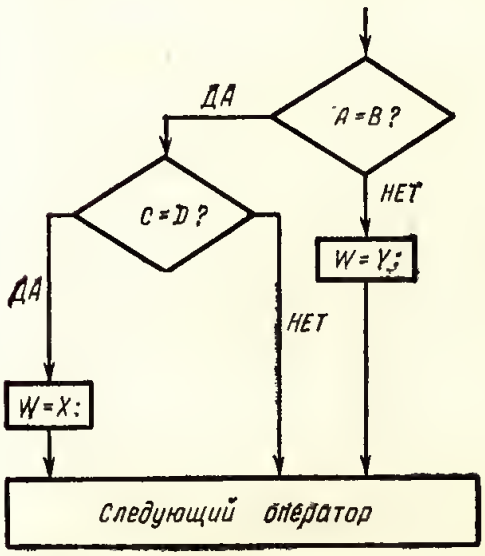
а)



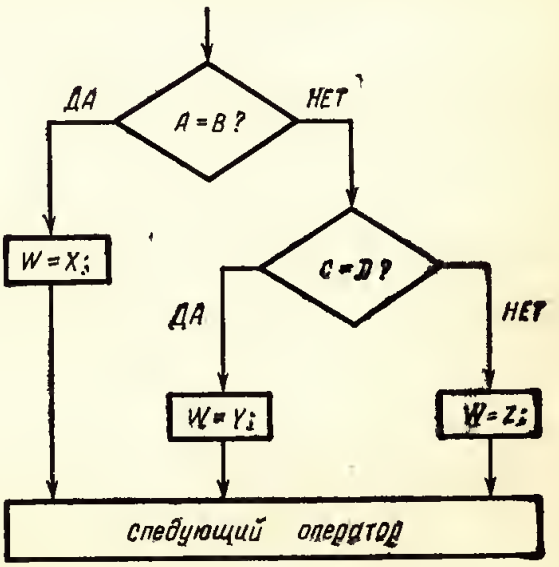
б)



в)



г)



11. а) Т; б) Т; в) не имеет значения, так как —¦D выполняется первым; г) Т; д) Т.

12.

```
ON ENDFILE (SYSIN) GO TO CONTINUE;
SUM_NEG, SUM_POS, NUM_ZERO=0;
READ: GET LIST (X);
      IF X<0 THEN SUM_NEG=SUM_NEG+X;
             ELSE IF X=0 THEN NUM_ZERO=NUM_ZERO+1;
                         EISE SUM_POS = SUM_POS+X;
      GO TO READ;
CONTINUE: _____;
```

13. Один из возможных ответов:

```
DECLARE (X (25,5) FIXED (3), AVE (25) FIXED (4, 1)) DECIMAL REAL;
```

14. Запись Фортранного типа может быть следующей:

```
PUT PAGE;
DO I=1 TO N;
PUT SKIP LIST (A(N-I+1));
END;
```

Более общий подход:

```
PUT PAGE;
DO I=N TO 1 BY -1;
PUT SKIP LIST (A(I));
END;
```

Глава 3

1. A DECIMAL FIXED REAL (6, 2)
   B DECIMAL FLOAT REAL (5)
   C (5) DECIMAL FLOAT REAL (6)
   D BINARY FLOAT REAL (21)
   E DECIMAL FLOAT REAL (6)
   F REAL DECIMAL FLOAT (6)
   G неверно; несовместимые атрибуты BINARY DECIMAL
   H REAL FLOAT DECIMAL (6)
   I (16) BINARY FIXED REAL (15,0)
   J DECIMAL FLOAT REAL (6)

2. а) INITIAL ((225)0)
   б) INITIAL (*, (14) ((15)1,*))
   в) INITIAL ((15) (0, 1, 0, 0, 2, (10)0))

3. В следующей записи символ b означает пробел:

Имя переменной	Присваиваемое значение
A	'Xbbbbbbbbb'
B	'A'BCbbbbbb'
C	'THISbISbAb'
X	'10101010'B
Y	'10100000'B
I	0032
J	4321
K	000000000101101B
Z	00005

```
4. DCL MARK (5) LABEL INIT (L1, L2, L3, L4, L5);
      .
      .
      .
   GO TO MARK (I);
```

5. *Задача 1*

```
DCL RADIUS FIXED (10, 4), VOLUME FLOAT (5);
```

Обратите внимание на то, что если бы VOLUME был объявлен по умолчанию с точностью (6), то он мог бы быть опущен из оператора DECLARE. В приведенном примере нет необходимости объявлять FLOAT, но так как описатель точности не может следовать непосредственно за именем идентификатора, ставится FLOAT, чтобы отличить точность от размерности.

Задача 2

```
DCL (A, B, C) FIXED (8, 3);
```

Задача 3

```
DCL (X, Y, COUNTER) FIXED (3), (XSUM YSUM XSQD, YSQD, XY) FIXED (8), (AVERA-
  GE_X, AVERAGE_Y) FIXED (7, 2);
```

Задача 4

```
DCL (NUMBER, MAXIMUM_NUMBER, MINIMUM_NUMBER) FIXED (15, 5);
```

Задача 5

```
DCL ((OLDTERM, OLDPI) INIT (0), (NEWTERM, NEWPI) INIT (3.OEO)) FLOAT (16), TERM
  FIXED (4) INIT (1), N FLOAT (16);
```

Задача 6

```
DCL A (20) FIXED (10) INIT (1, 1, (18)0);
```

Задача 7

```
DCL P FIXED (15, 2), I FIXED (11, 6), (Q, Y, F, L) FIXED (4);
```

Задача 8

```
DCL (A(501) INIT (1, 2, 3 (498)0), B(250), I INIT (2), T INIT (1)) FIXED (3);
```

Задача 9

```
DCL A(8) FIXED (6, 2) INIT (20., 10., 5., 1., .25, .1, .05, .01), (AMT_PD, DIFF, COST)
  FIXED (8, 2), (I, J, NT) FIXED (3);
```

Задача 10

```
DCL (INVALU, OUTVALU) FIXED (15, 2), EXPON FIXED (2);
```

Задача 11

```
DCL (DEGREES, MINUTES, SEC) FIXED (3), (SIN_OF_ANGLE, ANGLE1 ANGLE2, ANG-
  LE_INC) FIXED (10, 6);
```

Задача 12

```
DCL (RAND (500) INIT (1306, (499)0), CONSTANT, SUB, A(10) INIT ((2)1, 0, 1(2) 0,1(3)0)
  FIXED (4), N FIXED (3), VALUE1 FIXED (8), VALUE2 FIXED (7), C FIXED (1);
```

Задача 13

```
DCL (NUM, NUMBER (100)) FIXED (15, 5), (I, II, J, K) BINARY FIXED (4);
```

Заметьте, что как BINARY, так и FIXED необходимы.

Задача 14

```
DCL (AREA, POINT (500, 2)) FLOAT (8);
```

Глава 4

1. а) '00010'B
   б) '11111' B
   в) '10000' B
   г) '1' B
   д) '1' B
   е) '110111101'B
   ж) '1110110'B

2. а) 'ABCDEFGXYXYXY'
   б) 1001B
   в) OB
   г) 'EFABC'
   д) 'DEYYYYYFG'
   е) 'CABCABC'

3. Заметьте, что это не то же самое, что A, B = C + D. При необходимости для вычисления выражения C + D делается преобразование. Сравнение (=) B с C + D также может сопровождаться преобразованием. Результатом такого сравнения будет строка битов с длиной 1, которая затем будет присвоена переменной A.

4. а) '011'B
   б) '11111' B
   в) '01000' B

5. а)
```
I = INDEX (A,'');
J = INDEX (SUBSTR (A, I+1),'');
K = I+J;
```

Эти операторы могут быть объединены следующим образом:

```
K = INDEX(A, '')+INDEX (SUBSTR(A, INDEX (A, '')+1),'');
```

б) пусть Y описан описателем CHARACTER (1)

```
Y = SUBSTR (B, 10, 1);
SUBSTR (B, 10, 1) = SUBSTR (B, 32, 1);
SUBSTR (B, 32, 1) = Y;
```

6.
```
а) B = ¬B;
б) SUBSTR (B, 10, 1) = '1'B;
в) I = INDEX (B, '111' B);
```

7. а) $\begin{pmatrix} 3 & 3 & 4 \\ 4 & 1 & 6 \end{pmatrix}$

б) приведет к ошибке; A и C неодинаковой размерности.

в) $\begin{pmatrix} 4 & 4 & 5 \\ 5 & 2 & 10 \end{pmatrix}$

г) $\begin{pmatrix} 1 & 1 & 3 \\ 1 & 0 & 2 \end{pmatrix}$

д) $\begin{pmatrix} 2 & 3 & 2 \\ 4 & 2 & 6 \end{pmatrix}$ ж) (5 4 2) к) 864

е) $\begin{pmatrix} 2 & 4 \\ 3 & 2 \\ 3 & 6 \end{pmatrix}$ з) (5 3) л) $\begin{pmatrix} 6 & 9 \\ 9 & 26 \end{pmatrix}$

и) 3

8. а) Два раза; L принимает значения 1 и 4; б) семь раз, X принимает значения 10, 9, 7, 9.4, 9.1, 8.8, 8.5 и 8.2; в) четыре раза; I принимает значения 7, 5, 3 и 1; г) семь раз; K принимает значения 1, 3, 5, 7, 8, 9 и 10; д) если допустить, что LOOP является по умолчанию REAL FIXED BINARY (15,0), то это приведет к бесконечному повторению цикла. Начальное значение 3.2 усекается до 3, после чего цикл выполняется. Затем прибавляется приращение .3, давая 3.3, а 3.3 снова усекается до 3 перед вторым повторением цикла и т. д.

9. а)
```
LOOP: A=A+1;
        .
        .
        .
      IF A+B>6 THEN GO TO LOOP;
```

б)
```
                 I=1B;
              A: IF I=10 THEN DO;
                              оператор цикла
                              I=I+1B;
                              GO TO A;
                              END;
                 ELSE B:  DO;
                             оператор цикла
                             IF X=Y THEN GO TO B;
                             END;
```

в)
```
          X=10.1;
     LOOP; X=X-.1;
          IF X<9. THEN GO TO EXIT;
          первая группа операторов
          I=0B;
IN__LOOP: I=I+1B;
          IF I>11B THEN GO TO LOOP;
          IF I=1B THEN CHAR='ONE ';
                  ELSE IF I=10B THEN CHAR='TWO ';
                               ELSE CHAR='THREE';
          вторая группа операторов
          GO TO IN__LOOP;
EXIT:_____;
```

10.
```
P__10: PROCEDURE OPTIONS (MAIN);
       DCL ANS BIT (60);
       ON ENDFILE (SYSIN) GO TO PRINT;
       I=0B; J=0B;
   IN: GET LIST (ANS);
       I=I+1B;
       IF SUBSTR (ANS, 10, 1) & ¬ SUBSTR (ANS, 13,3) & ((SUBSTR (ANS, 25, 1)
       & SUBSTR (ANS, 26, 1)) | (¬ SUBSTR (ANS, 25, 1) & SUBSTR (ANS, 26, 1)))
       THEN J=J+1B;
       GO TO IN;
```

```
PRINT:  PUT LIST (I, J);
        END P___10;
```

Оператор IF в этой программе может быть записан:

```
IF SUBSTR (ANS, 10, 1) & —¬ SUBSTR (ANS, 13, 3) &
   BOOL (SUBSTR (ANS, 25, 1), SUBSTR (ANS, 26, 1), '0110'B)
   THEN J=J+1B;
```

Глава 5

1. b означает пробел:

```
   —764.870                          —7.649E+02
bbbb—764                            b—7.65E+02
   —7648.700                        невозможно; ширина поля должна быть 10
  bb—76.487                          —7.649E+02
   b—764.9                           —7.64870E+02
 bbb—7649                          bbbb—764.87E+00
```

2. а) PUT EDIT (A) (PAGE, 16 (F (6,2), SKIP));
   б) PUT EDIT (A) (PAGE, 16 (F (6,2), SKIP (2)));
   в) PUT EDIT ((A (I), B (I), C (I) DO I=1 TO 64)) (PAGE, 32 (3 F (20,2), SKIP));

г) Пусть HEADING — строка символов длиной 30 и в ней хранится

```
'bbbARRAYbAbbbARRAYbBbbbARRAYbC'
PUT EDIT ((HEADING, (A (J+1), B (J+1), C (J+1) DO I=1 TO 32) DO J=0,32))
          (PAGE, A (30), SKIP (3), 32 (3 F (10,2), SKIP));
```

3. а) GET LIST ((A (I), B (I) DO I=1 TO 64));
   б) GET EDIT ((A (I), B (I) DO I=1 TO 64)) (F (10,2));

Нельзя полагать, что пробелы будут разделять элементы данных, как это сделано в примере а).

в) GET EDIT (A, B) (F (10, 2));

4. а) вероятно, самое легкое — оставить на карте по крайней мере один пробел после 12.76 и затем отперфорировать A (5,9) = 13.2, A (7,18) = 15.75; и считывать эти значения оператором GET DATA;
   б) воспользуйтесь оператором

```
GET LIST ((A (I, I) DO I=1 TO 20));
```

и отпечатайте данные в произвольной форме, разделив их друг от друга по крайней мере одним пробелом.

5. а) PUT EDIT ((A(I) DO I = 1 TO 30)) (SKIP, F (7,2)); Заметьте, что список данных не может состоять только из A, так как массив A имел размерность 50.

б) PUT SKIP EDIT (((A (I,J) DO J=1 TO 30) DO I=1 TO 10)) (F (7,2), SKIP);

Список данных также не может состоять только из A.

в) PUT SKIP EDIT (((A (I, J) DO I=1 TO 10) DO J=1 TO 30)) (F (7,2), SKIP);

6. PUT EDIT ('PL/1', 'PAGE 1') (PAGE, LINE (3), X (58), A (56), A);

Удовлетворительными могут также быть следующие два списка форматов:

```
(PAGE, SKIP (2), X (58), A, X (52), A); или (PAGE, SKIP (2), COLUMN (59), A,
  COLUMN (115), A);
```

```
7. PUT EDIT (('PAGE', N, ((A (I,30* (N-1)+J) DO I=1 TO 5) DO J=1 TO 30) DO N=1
            TO 20)) (PAGE, COLUMN (114), F (3), SKIP (8), 150 F (24, 4));
8. J=0, K=3, L=2;
9. PUT PAGE;
                DO L=15, 17,35;
                PUT EDIT ((M DO I=1 TO 4)) (LINE (L), 4 A (30));
                END;
10. PUT PAGE EDIT (A) (5 F (10), SKIP (6));
```

```
11. PR__11; PROCEDURE OPTIONS (MAIN);
            ON ENDFILE (SYSIN) GO TO PRINT;
            SUM=0;
    READ;   GET LIST (X);
            IF X>0 THEN SUM=SUM+X;
            GO TO READ;
    PRINT;  PUT LIST (SUM);
            END PR__11;
```

Если вы предпочли оператор GET EDIT, то почему?

12. Возможны два решения:

```
a) IN; GET LIST (I);
      IF I=1 THEN DO;
                  GET LIST (X);
                      .
                      .
                      .
                  END;
            ELSE DO;
                  GET EDIT (CHAR) (COLUMN (15), A (36));
                      .
                      .
                      .
                  END;
      GO TO IN;
```

Заметьте , что для считывания строки символов необходимо применить EDIT, так как в перфокарте данных не было кавычек.

```
б) DECLARE STOP CHAR (80);
   IN; GET EDIT (STOR) (A (80));
   IF SUBSTR (STOR, 1,1)='I' THEN DO;
         GET STRING (STOR) EDIT (X) (X (9), F (11, 12));
                  .
                  .
                  .
         END;
                     ELSE DO;
         GET STRING (STOR) EDIT (CHAR) (X (14), A (36));
                  .
                  .
                  .
         END;
   GO TO IN;
```

```
13. I=4, J=8, Z=27.86, X=4.3, Y=8.72, A=31, B=46.721, C=3, D=15.7
14. а) (F (14), X (4), A (8), F (10,2), X (15), A, F, (7,2));
    б) (SKIP, X (5), 2 E (20,6), E (25,6), SKIP (2), F (6,1), F (14))
```

Глава 6

1. а) неправильно; значение N неизвестно до времени выполнения программы; б) то же, что в примере а); в) правильно, печать 3,1. 2.3. г) неправильно; д) неправильно; переменная A на предпоследней строке имеет идентификатор, отличный от идентификатора переменной A в процедуре XYZ; е) правильно; печать 31.2.3.; ж) правильно; печать 1.2.3.3

2. а)

A	B, F, G
B	C, E, F, G
C	E, F, G
D	E, F, G
E	C, F, G
F	B, G
G	H
H	нет

б) идентификатор LAB должен быть определен ко времени выполнения оператора GO TO LAB; Если меткой LAB помечен только один оператор, он может быть в любом активном блоке, а те четыре, которые, как нам известно, являются активными, — это A, B, C и G. Если в активных блоках имелись два оператора с меткой LAB, то управление передается оператору с этой меткой в блоке, в котором объявление LAB появилось ранее всего (либо явное, либо неявное).

в) для того чтобы G стало именем входа в процедуру F, оно должно быть объявлено явно включением G ENTRY в оператор описания;

г) в этой программе три различных Y. Область действия первого — A, B, C, D и E; область действия второго — F; область действия третьего — G и H;

д) в этой программе два различных X. Область действия первого — B, C, D, E, G и H; область действия второго — F;

е) к моменту выполнения этого оператора активными будут следующие блоки: A, B, E, C и D. Так как блок E активен, то его нельзя вызвать, если только он не описан описателем RECURSIVE;

ж) обращение к G возможно, так как он не активен и представляет собой внешнюю процедуру;

з) нет. Обращение к процедуре H в настоящее время невозможно. Оно стало бы возможным при использовании точки входа в процедуру G. Однако необходимо очень тщательно следить за тем, чтобы в результате применения добавочных операторов при вызове процедуры G в какое-либо другое время не вызывалась бы и процедура H. Решение могло бы быть следующим:

```
G:   PROCEDURE;
     DCL X EXTERNAL, Y FIXED;
                .
                .
                .
     GO TO XYZ;.
GP:  ENTPY;
     CALL H;
     RETURN;
```

```
H;   PROCEDURE;
       .
       .
       .
     END H;
XYZ: ____________________
       .
       .
       .
     END G;
```

и) по умолчанию H вырабатывает значение переменной, имеющей описатели DEC FLOAT REAL (6). Последние не совпадают с описателями переменной, полученной в результате выполнения (4* M — N), а именно BIN FIXED REAL (15,0). Это положение можно исправить, изменив оператор процедуры в H: PROCEDURE (M, N) BIN FIXED; и добавив оператор DCL H RETURNS (BIN FIXED); к процедуре G.

3. 0, 7, 6.

Процедуры FOF и GOF представляют определение следующих процедур:

$$\begin{cases} f(0) = 0 \\ \text{If } X \neq 0: \\ f(X) = x + g(x) \\ g(x) = 4 - f(x-1) \end{cases}$$

4. A активизируется с начала выполнения программы и остается активным все время ее выполнения. Оператор (2) объявляет X управляемой переменной размерности 100 и она может храниться или освобождать память в A или B. Переменная Y хранится в памяти во время всего выполнения программы, но она неприменима в блоке C. (Заметьте, что это будет справедливо для Y, даже если он не описан описателем STATIC, так как A остается активным.) Для переменной Z выделяется память, и она может быть использована только в B (так как B — внутренняя процедура) и в C (так как она объявлена описателем EXTERNAL).

Операторы (3 — 6) отводят для X 100 ячеек памяти, считывают 100 значений X, подсчитывают их сумму, хранят этот результат в Z, а затем освобождают 100 ячеек X. Оператор (7) вызывает B.

Ко времени вызова B определены все идентификаторы процедуры A, за исключением X, которой в это время не было в памяти, и L. Оператор (14) (в A) объявляет L константой типа метки, в то время как оператор (9) (в B) объявляет L переменной типа метки.

Оператор IF в (10) вызовет либо внешнюю процедуру C, либо произведет приращение I на 1 и продолжит выполнение процедуры B. При дальнейшем выполнении программы оператор (12) передаст управление оператору, следующему за оператором CALL в процедуре A. В данном случае это будет оператор, следующий непосредственно за оператором (12). Все идентификаторы, определенные только в B (такие, как L, I, J), далее будут недоступными. Если выполняется оператор CALL C (I, L), то имена идентификаторов I и L становятся M и N в процедуре C, к которой произведено обращение. Идентификатор I

оператора (18) отличается от I оператора (10). При выполнении GO TO N в операторе (21) управление передается оператору с меткой L в операторе (11). Освобождаются все области памяти, известные только в C, а выполнение программы продолжается оператором (11).

После выполнения оператора (12) управление передается в A, как это имело место в ранее приведенном примере, когда выполнялась конструкция ELSE в операторе (10). Выполнение процедуры A продолжается до оператора (13), после чего снова может быть вызвана процедура B. И наконец, когда управление передается оператору (15), выполнение программы заканчивается.

Глава 7

```
1. а) DCL B (100) DEFINED A (3, 1SUB);
   б) DCL B (100) DEFINED A (1SUB, 21);
   в) DCL B (100) DEFINED (A) (1SUB, 1SUB);
   г) DCL B (50) DEFINED A (7.2* 1SUB);
   д) DCL B (7) DEFINED A (2** (1SUB-1)), 2** (1SUB-1));
```

2. а) строка 'ABCDEFGH' хранится в A; B относится к 'AB'; C относится к 'DEF'; последняя часть неправильная и вызовет сообщение об ошибке, так как C объявлена переменной. Последнюю часть можно выполнить при

```
D CHAR (2) DEFINED A POSITION (4)
```

б) Y (1), ..., Y (50) относится к A (50), ..., A (1)

в) A (I) представляет *i*-ю ячейку памяти в массиве B, в том порядке, в каком обычно распределяется память, причем первый индекс изменяется медленнее остальных;

г) в данном примере B (0), B (1), ..., B (9) относится к A (0), A (1), A (3), A (5); A (9), A (8), A (6), A (7), A (4), A (2);

д) массив B заполняется 18 элементами массива A, размерность которого равна 10. Первая строка массива B состоит из первого и второго элементов A, вторая строка массива B состоит из второго и третьего элементов массива A и т. д.

3. а) 3456.0
   б) b1234
   в) — 1234
   г) 12/34/56
   д) b102b3
   е) b00123
   ж) bbb123
   з) bbb123$
   и) $ 1,234.56
   к) bbb $ 12.34DB
   л) ** 1000
   м) 12b3
   н) 3400

о) 2340
п) — 12.340
р) 12.34

4. а) 1) 14, 2) 1, 3) 4, 4) идентификатора в программе нет, 5) идентификатора в программе нет, 6) идентификатора в программе нет, 7) 1.

б) 1) X=2* TOTAL___RECORD. FIELD___GOALS. MADE+TOTAL___RECORD. FOULS. MADE;
2) TOTAL___RECORD=TOTAL___RECORD+LAST___GAME___RECORD;
SEASON___RECORD=SEASON___RECORD+LAST___GAME___RECORD;

5. а) все A и B, но не C;

б) добавьте префикс (NOCONVERSION) к оператору B: BEGIN;

в) (CONVERSION, NOZERODIVIDE): A: PROCEDURE;
.
.
.
(ZERODIVIDE): B: BEGIN;
.
.
.
END B;
.
.
.
END A;
(NOZERODIVIDE): C: PROCEDURE;
.
.
.
END C;

г) NOUNDERFLOW применяется только к Z/Y*Z < 3*Y. Эта ситуация не применяется к конструкциям THEN или ELSE.

д) (SIZE): DO WHILE (X — Y > Z); не выполнимо, так как это применимо только к X — Y > Z. Префикс (SIZE) необходимо добавлять к двум операторам, следующим за DO.

е) ON CONVERSION GO TO CON;

ОГЛАВЛЕНИЕ

СКОТТ Р.,
СОНДАК Н.

ПЛ/1 ДЛЯ ПРОГРАММИСТОВ

Редактор *Е. В. Крестьянинова*

Мл. редактор *О. Б. Степанченко*

Техи. редактор *Р. Н. Феоктистова*

Корректоры *Т. М. Васильева, Г. М. Брызгунова*

Худ. редактор *Н. А. Володина*

Переплет художиика *В. Л. Николаева*

ИБ № 254

Сдаио в иабор 3/XI 1976 г.
Подписано к печати 4/III 1977 г.
Формат бумаги 60×90¹/₁₆ Бумага № 2 Объем 14,0 печ. л.
Уч.-изд. л. 15,33 Усл. п. л. 14,0 Тираж 76 000 экз.
(Тематич. плаи 1977 г. № 108)
Заказ 1278 Цеиа 1 р. 08 к.

Издательство «Статистика», Москва, ул. Кирова, 39.

Московская типография № 4 Союзполиграфпрома
при Государствеином комитете Совета Министров СССР
по делам издательств, полиграфии и книжиой торговли,
Москва, И-41, Б. Переяславская ул., д. 46

Р. СКОПП
Н. СОНДАК

ПЛ/1 ДЛЯ ПРОГРАММИСТОВ